Department of Defense

High Level Architecture Run-Time Infrastructure Programmer's Guide

Version 1.0

15 May 1997

DMSO MITRE SAIC An Employee-Owned Company VTC Virtual Technology Corporation Software and Systems Engineering Professionals

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

1. Overview

This document is designed to aid developers of distributed models, simulations, analysis tools, and other applications to gain a better understanding of the High Level Architecture (HLA) and the Run-Time Infrastructure (RTI). This document is not intended to describe the concepts of modeling and simulation or the HLA, but to provide a starting point for an experienced simulation developer to use the RTI. To best understand the RTI and this programmer's guide, one should first have a familiarity with the higher-level components and issues that comprise the Department of Defense (DoD) Modeling and Simulation Master Plan. This plan defines the objective of developing a Common Technical Framework for Modeling and Simulation (M&S), consisting of three components for simulation development and interaction: the High Level Architecture (HLA), Conceptual Model of the Mission Space (CMMS), and Data Standardization (DS), all of which are briefly discussed below. More detailed information can be found at the Defense Modeling and Simulation Office's (DMSO) home page at http://www.dmso.mil.

This section provides a brief overview and recap of the HLA, CMMS, and DS. Section 2 provides a description of the system architecture for the 1.0 version of the RTI. Section 3 provides a description of the 1.0 C++ classes that implement the HLA 1.1 Interface Specification. Section 4 contains a programming tutorial which demonstrates the use of the RTI services. Section 5 highlights some of the common problems which may be encountered when configuring and using the RTI.

1.1 High Level Architecture

The HLA establishes a common high-level simulation architecture to facilitate the interoperability of all types of models and simulations among themselves and with C4I systems, as well as to facilitate the reuse of M&S components.

HLA is defined by the following three components:

- HLA rules: These describe the responsibilities of federates and of the Run-Time Infrastructure (RTI) in HLA federations. For example, federations must have a Federation Object Model (FOM) defined using the Object Model Template format, which defines data exchanged by the RTI. Similarly, each federate has an HLA Simulation Object Model (SOM), defining the attributes and objects the federate updates, reflects, sends, receives, and transfers ownership.
- Interface specification: This defines the six service groups between the RTI and the federates. These service groups are: (1) Federation management: the creation, dynamic control, modification, and deletion of a federation execution; (2) Declaration management: the intent to publish and subscribe to object attributes and interactions; (3) Object management: the creation and deletion of object instances and the communication of attributes and interactions; (4) Ownership management: the transfer of ownership of object attributes.; (5) Time management: the coordination of advances in simulation time; and (6) Data distribution management: the support of efficient data routing.
- Object Model Template: This is the common method for representing HLA Object Model information, for example, information found in the Federation Object Model (FOM) and Simulation Object Model (SOM).

Run-Time Infrastructure

The RTI is a set of software components that implement the services specified by the HLA Interface Specification. The RTI is the general purpose software that provides the common interface services for the execution of an HLA federation. The RTI provides these services to federates in a way that is analogous to how a distributed operating system provides services to applications.

For more detailed information about HLA, please access documentation available at http://www.dmso.mil/projects/hla.

1.2 Conceptual Model of the Mission Space

A Conceptual Model of the Mission Space (CMMS) is a first abstraction of the real world, which serves as a common framework for knowledge acquisition with validated, relevant actions and interactions organized by specific task and entity/organization. It is a simulation independent hierarchical description of actions and interactions among the various entities associated with a particular mission area.

Thus, conceptual models of the mission space provide simulation developers with a common baseline for constructing consistent and authoritative M&S representations. The primary purpose of CMMS is to facilitate interoperability and reuse of simulation components, particularly among DoD simulation developments, by sharing common, authoritative information between DoD simulations. CMMS will provide a meta-model of fundamental knowledge about military operations. The CMMS System will capture and store this knowledge, and make it easily accessible to simulation developers and users.

The mission space structure, tools and resources will provide both an overarching framework and access to the necessary data and detail to permit development of consistent, interoperable, and authoritative representations of the environment, systems, and human behavior in DoD simulation systems.

For more detailed information about CMMS, please access documentation available at http://www.dmso.mil/projects/cmms.

1.3 Data Standardization

The data standardization program seeks to facilitate reuse, interoperability, and data sharing among models, simulations, and C4I systems by establishing policies, procedures, and methodologies for data requirements, standards, sources, security, and verification, validation, and certification.

The primary products of the data standardization program are: (1) Common Semantics and Syntax (CSS), which define common lexicons, dictionaries, taxonomies, and tools for data elements; and (2) Data Interchange Formats (DIF), the physical structures (BNF, SQL) used by programmers to actually interchange data.

Other supporting data standardization products are: (1) Authoritative Data Sources (ADS), the primary means for identifying data for reuse; (2) Data Quality (DQ) practices, a body of VV&A/C guidelines; and (3) Data Security (DS) practices, the policies pertaining to data protection and release.

For more detailed information about data standardization, please access documentation available at http://www.dmso.mil/projects/ds.

2. RTI 1.0 System Architecture

RTI 1.0 is a distributed system comprised of two global processes, the RTI Executive (rtiexec) and the Federation Executive (fedex), and a library that is linked into each federate. The rtiexec is a well-known process that manages the creation and destruction of federation executions. The fedex is a global process per federation execution that manages the joining and resigning of federates in an execution. The linkable library provides the federate developer with the interface and implementation of a majority of the HLA 1.1 services. The HLA 1.1 services are performed via communication between the rtiexec, the fedex, and the federates utilizing socket-based reliable and best effort inter-process communication (IPC). See Figure 2-1: RTI 1.0 System Architecture.

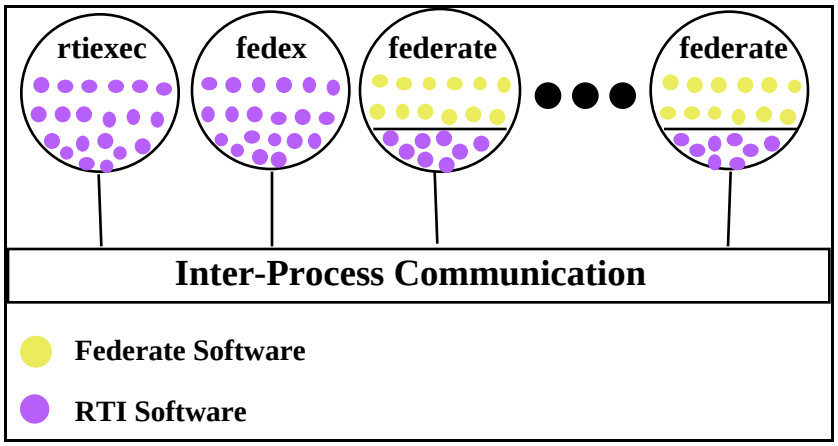

Figure 2-1: RTI 1.0 System Architecture

2.1 Run Time Infrastructure (RTI) Executive

The RTI Executive (rtiexec) is the well-known global process that each federate will communicate with during its initialization of the RTI components. The RTI Executive's primary purpose is to manage the creation and destruction of federation executions. The management of federation executions includes ensuring that each federation execution has a unique name, providing a joining federate with the handle (hostname and port) to an existing federation execution, and providing a unique multicast group for each federation execution to communicate best-effort data.

The rtiexec executable has a console interface that provides access to commands that can be used to help manage the current list of federation executions. The set of commands currently available is described in Table 2-1: RTI Executive Console Commands.

Table 2-1: RTI Executive Console Commands

Command Usage	Description
help	Lists the available commands and their usage.
List	Lists the set of federation executions that are currently registered with the rtiexec .
ref <fedName>	Returns the host and port for the specified federation execution.
remove <fedName>	Removes a federation execution from the rtiexec.
quit	Exits the rtiexec process.

Note: The location of the rtiexec (host and port) is specified in the RTI.rid file. Federates will not be able to communicate with the rtiexec process if the RTI_EXEC_HOST and RTI_EXEC_PORT values are incorrectly specified in the RTI.rid file. For more details on the RTI.rid file, see Section 2.4.1, Run-time Initialization Data (RID).

2.2 Federation Executive

The Federation Executive (fedex) is a global process per federation execution that manages the joining and resigning of federates and performs distribution of all reliable UpdateAttributeValues, SendInteractions, and all RTI internal control messages. The fedex process is created (fork/execvp'd) by the first federate to successfully invoke the Create Federation service for a given Federation Execution name. During initialization, the fedex process communicates with the rtiexec to register itself and to request a multicast address for its federations' best-effort communications. When a federate invokes the Join Federation Execution service, the fedex provides the federate with an enumerated handle and a multicast address to use for broadcasting best-effort communication. (Note: Only Update Attribute Value and Send Interaction services can be communicated using best-effort transport.) For reliable Update Attribute Value and Send Interaction services, the fedex acts as an information exploder by receiving the reliable communication from the sending federate (point-to-point using TCP) and then iterating through all other federates sending the reliable communication, as indicated in Figure 2-2: Reliable Update Attribute Value & Send Interaction Communication.

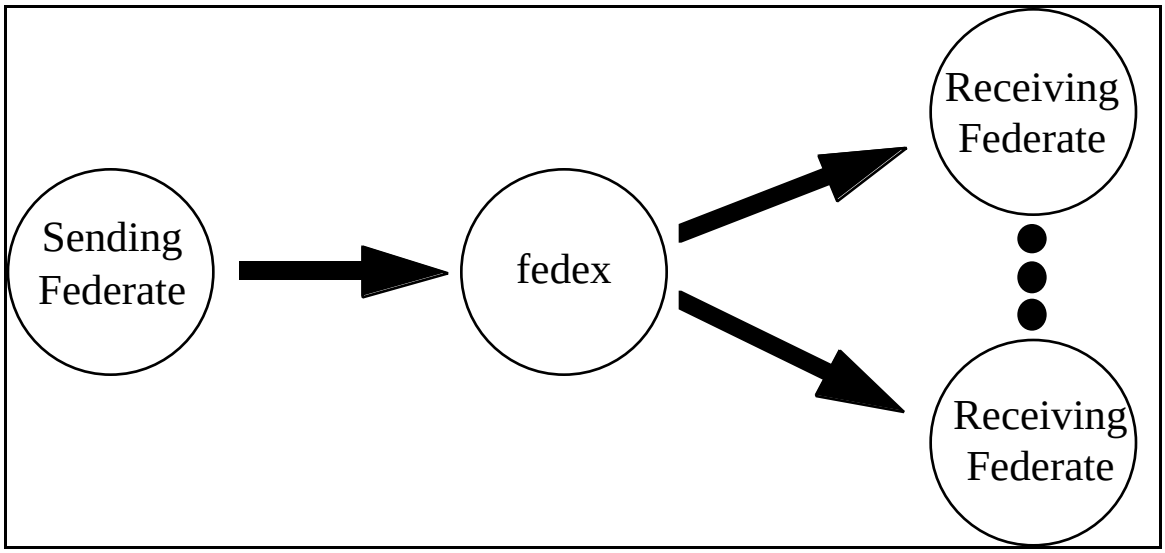

Figure 2-2: Reliable Update Attribute Value & Send Interaction Communication

The fedex executable has a console interface that provides access to commands that can be used to help manage the current list of federates. The set of commands currently available is described in Table 2-2: Federation Executive Console Commands.

Table 2-2: Federation Executive Console Commands

Command Usage	Description
help	Lists the available commands and their usage.
list	Lists the set of federates that are currently joined to the federation execution.
ref <handle>	Prints the hostname and port for the federate handle.

Table 2-2: Federation Executive Console Commands

Command Usage	Description
remove <handle>	Removes a federate from the federation execution.
trace	Toggles trace mode on or off.
quit	Exits the fedex process.

2.3 C++ Library (libRTI)

The C++ library (libRTI) implements the interface to the HLA 1.1 services. Each federate in a federation execution will utilize the C++ library to invoke HLA services. These services are performed via communication with the rtiexec, the fedex, and other federates in the federation execution. This section will provide a high level description of the C++ Application Programming Interface (API), memory management conventions, and issues regarding data marshaling.

2.3.1 C++ Application Programming Interface (API)

The two classes that provide the interface between the federate and the RTI are the RTIambassador and FederateAmbassador. The RTIambassador class defines and implements the interface that is used by the federate to communicate with the RTI. The FederateAmbassador class defines the interface the RTI will use to communicate with the federate. The FederateAmbassador class is an abstract base class that the federate developer must implement (sub-class and define the methods) in order to successfully compile a federate with the RTI. Figure 2-3: RTI 1.0 C++ Interface, depicts the two classes that provide the interface between the federate and the RTI.

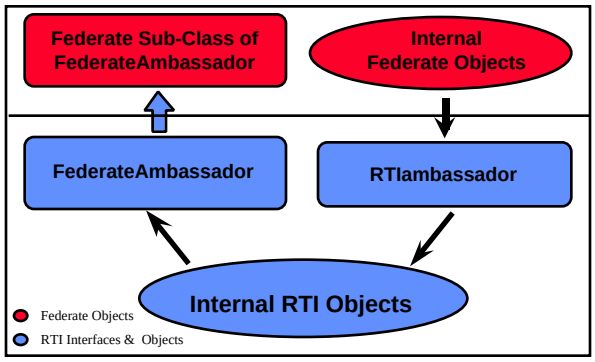

Figure 2-3: RTI 1.0 C++ Interface

2.3.2 Memory Allocation Conventions

The general convention for memory allocation/deallocation is that the application developer must deallocate any memory that the developer allocated on the heap (using the new function). In addition, some RTI 1.0 class methods allocate memory on the heap and pass it to the application. Each of the methods in the RTI 1.0 include files have been annotated with the conventions described in Table 2-3: Memory Allocation Conventions, which follows.

Table 2-3: Memory Allocation Conventions

Code	Convention Description	Allocator	Deallocator
C1	In parameter by value.	None	None
C2	Out parameter by reference.	None	None
C3	Function return by value.	None	None
C4	In parameter by const reference. Caller provides memory. Caller may free memory or overwrite it upon completion of the call. Callee must copy during the call anything it wishes to save beyond completion of the call. Parameter type must define const accessor methods.	Caller	Caller
C5	Out parameter by reference. Caller provides reference to object. Callee constructs an instance on the heap (new) and returns. The caller destroys the instance (delete) at its leisure.	Callee	Caller
C6	Function return by reference. Callee constructs an instance on the heap (new) and returns a reference. The caller destroys the instance (delete) at its leisure.	Callee	Caller

2.3.3 Data Marshaling

Communicating federation data across heterogeneous platforms (e.g., Sun, SGI, HP, IBM, Windows NT) requires a policy for the conversion of data between platforms. A commonly used policy is to convert data between the platform specific data representations and a platform independent representation (usually called network representation). This conversion requires knowledge about the data types to be converted. Since the RTI 1.0 does not know the types of a federate's attributes and parameters, it can not perform this conversion.

2.3.4 Internal Software Design

RTI 1.0 provides a procedural interface to the HLA 1.1 services through the RTIambassador and FederateAmbassador classes. Each of the services is executed by the federate within the federate's thread of control. Figure 2-4: RTI 1.0 Internal Architecture depicts the high level objects within the RTI 1.0 system and annotates the objects with a description of their functionality.

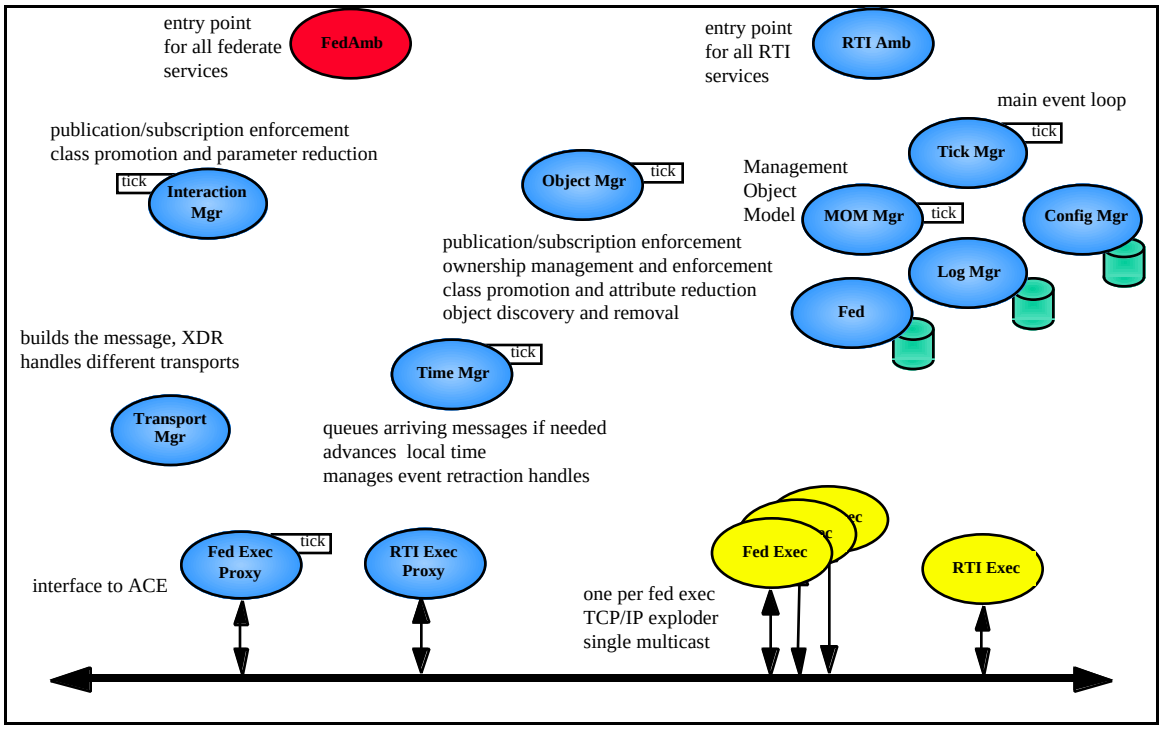

Figure 2-4: RTI 1.0 Internal Architecture

2.3.4.1 Flow of Control

An application provides the flow of control to the RTI by using the RTIambassador::tick() method. During the tick() method, the RTI performs many operations that are transparent to the application as well as the operations which invoke FederateAmbassador methods (similar to application callbacks). An example federate event loop is shown in Figure 2-5: Federate Event Loop Example.

```
while ( RTI::RTI_TRUE ) // My event loop lasts forever
{
   try
   {
      timeAdvGrant = RTI::RTI_FALSE;
      rtiAmb.timeAdvanceRequest(currentTime + timeStep);
   }
   // Should catch exceptions here

   while (!timeAdvGrant)
   {
      //---------------------------------------------------
      // Tick will turn control over to the RTI so that it can
      // process an event.  This may cause an invocation of one
      // of the federateAmbassadorServices methods.
      //
      // Be sure not to invoke the RTIambassadorServices from the
      // federateAmbassadorServices; otherwise, a ConcurrentAccess
      // exception will be thrown.
      //---------------------------------------------------
      int eventsToProcess = 1;
```

```
      // tick in a loop to get all events currently queued up
      while ( eventsToProcess )
      {
         eventsToProcess = rtiAmb.tick();
      }
   }

   //-----------------------------------------------------
   // If a time advance grant occurred and we have been given
   // permission to advance in time then calculate my next state.
   //-----------------------------------------------------
   if (grantTime > currentTime)
   {
      //-----------------------------------------------------
      // Do some simulation stuff here and set my new time...
      //-----------------------------------------------------
      currentTime = grantTime;
   }
}
```

Figure 2-5: Federate Event Loop Example

The RTI is a distributed system with components located throughout the federation. Each RTI component (linked into each federate) must perform synchronization operations with the other RTI components to allow a federation to progress in time, handle ownership management, join federations, and update Management Object Model (MOM) state. It is important to invoke the tick method regularly (not just when you want data) since many internal systems are invoked by RTIambassador::tick(). See Figure 2-4: RTI 1.0 Internal Architecture for a depiction of internal RTI components that are provided flow of control during a RTIambassador::tick().

2.3.4.2 Thread Model

The RTI 1.0 is intended to operate within a single thread (the main thread) of an application. However, the RTI library has been compiled with the -mt flag and has been tested in applications that instantiate the RTI objects in a thread other than the main process thread. The RTI is not reentrant and enforces this by throwing the exception RTI::ConcurrentAccessAttempted - this occurs when an RTIambassador method is invoked while another RTIambassador method has not yet completed (this includes the tick() method). However, the following RTI support services are reentrant and can be invoked within the scope of a FederateAmbassador method: getObjectClassName, getObjectClassHandle, getAttributeName, getAttributeHandle, getInteractionClassName, getInteractionClassHandle, getParameterName, and getParameterHandle.

2.4 Configuration and Input Files

The RTI 1.0 requires that a federation execution data (FED) file and a run-time initialization data (RID) file exist to run a federation execution. The FED file contains the data organization agreed upon in the FOM along with default transport and ordering information for object attributes and interaction class data. The RID file contains configuration parameters that the 1.0 uses to fine-tune and modify its system configuration at run-time. 1.0 expects to find the RID and FED files in the directory specified by the RTI_CONFIG environment variable. The files are named RTI.rid and <Federation Name>.fed, respectively.

2.4.1 Run-time Initialization Data (RID)

The Runtime Initialization Data (RID) is RTI implementation specific information used to fine-tune RTI behavior and system configuration at run-time. In 1.0, this file is parsed by the fedex and each federate (libRTI). An example RTI.rid file is shown in Figure 2-6: Example RTI.rid file.

```
###############################################################################
# FILE   : RTI.rid
# PURPOSE: This file is the main configuration file for the RTI.
###############################################################################


###############################################################################
# VARIABLE: ATTRIBUTE_RELEASE_TRIES
# UNITS   : Positive integer
# PURPOSE : To specify the number of times the RTI Ambassador should tick the
#           Federation Execution in an attempt to release the attributes of a
#           resigning federate.
###############################################################################
ATTRIBUTE_RELEASE_TRIES     2

###############################################################################
# VARIABLE: AUTO_TICK_PERIOD
# UNITS   : Positive double
# PURPOSE : To specify the automatic tick period during join and resign.
###############################################################################
AUTO_TICK_PERIOD    1.0

###############################################################################
# VARIABLE: BEST_EFFORT_PORT
# UNITS   : Positive integer
# PURPOSE : To specify the port number on which best-effort multicast
#           addressing will be attempted.
###############################################################################
BEST_EFFORT_PORT    18134

###############################################################################
# VARIABLE: MAX_HANDLE_VALUE_PAIRS
# UNITS   : Positive integer
# PURPOSE : To specify the maximum number of attribute handle value pairs
#           allowed in an attribute handle value pair set.
###############################################################################
MAX_HANDLE_VALUE_PAIRS    5000

###############################################################################
# VARIABLE: MAX_OBJECTS_PER_FEDERATE
# UNITS   : Positive integer
# PURPOSE : To specify the maximum number of objects a federate may know about.
###############################################################################
MAX_OBJECTS_PER_FEDERATE  100000

###############################################################################
# VARIABLE: MOM_TIME_RESOLUTION
# UNITS   : Positive integer
# PURPOSE : To specify the tick interval for checking the wall clock to see if
#           it is time to send a report.
###############################################################################
MOM_TIME_RESOLUTION     10

###############################################################################
# VARIABLE: DELETED_OBJECT_DURATION
# UNITS   : Positive integer
# PURPOSE : To specify the number of tick to wait before attempting to
```

```
#           remove DELETED objects from the object database.
###############################################################################
DELETED_OBJECT_DURATION  10

###############################################################################
# VARIABLE: RTI_EXEC_HOST
# UNITS   : Character string
# PURPOSE : To specify the hostname of the machine on which the RTI Executive
#      process is executing.
###############################################################################
RTI_EXEC_HOST    localhost

###############################################################################
# VARIABLE: RTI_EXEC_PORT
# UNITS   : Positive integer
# PURPOSE : To specify the port number on which the RTI Executive process is
#      listening for connections.
###############################################################################
RTI_EXEC_PORT    18134

###############################################################################
# VARIABLE: TIME_TRACE
# UNITS   : Boolean
# PURPOSE : To specify whether or not to use tracing in the Time Manager.
###############################################################################
TIME_TRACE    OFF

###############################################################################
# VARIABLE: FEDEX_TIMEOUT
# UNITES  : Positive Integer
# Purpose : To specify how long the FedEx should wait for input before handing
#           out orphaned objects to federates
###############################################################################
FEDEX_TIMEOUT    120
```

Figure 2-6: Example RTI.rid file

2.4.2 Federation Execution Data (FED)

The federation execution data (FED) describes the information in the FOM that the RTI needs to properly handle the global federation name space. The information contained in the FED describes the inheritance structure of object and interaction classes, as well as the attributes and parameters at each level in the respective class hierarchy.

When a federate joins a federation execution, components within the RTI library will read the <FederationExecutionName>.fed file that is found in the directory specified by the RTI_CONFIG environment variable. This file must contain each of the object and interaction classes and attributes and parameters that will be used in a federation execution. The FED syntax is specified in a pseudo-lisp format of nested lists of tokens. The FED file syntax is shown in Figure 2-7: Federation Execution Data (FED) file syntax.

```
;; Comments are any text after a semicolon.
;; basic syntax example
;; possible <transportation> = FED_RELIABLE,
;;                             FED_BEST_EFFORT
;;
;; possible <ordering> = FED_RECEIVE,
```

```
;;                        FED_TIMESTAMP
;;

(fed
;; object, class, and attribute definitions follow

   (objects
      ( class <name>
         (attribute <name> <transportation> <ordering>)
         (attribute <name> <transportation> <ordering>)
;;   ... any other attributes must come before any subclasses for same level
         (class <name>
            (attribute <name> <transportation> <ordering>)
            (attribute <name> <transportation> <ordering>)
         )
      )
   )

;; interactions, class, and parameter definitions follow

   (interactions
      (class <name> <transportation> <ordering>
         (parameter <name>)
         (parameter <name>)
;;   ... any other parameters must come before any subclasses for the same level
         (class <name> <transportation> <ordering>
            (parameter <name>)
            (parameter <name>)
         )
      )
   )
) ; end of fed
```

Figure 2-7: Federation Execution Data (FED) file syntax

2.4.2.1 Management Object Model (MOM)

The Management Object Model (MOM) defines the set of object classes and interaction classes used for RTI and federation specific management and monitoring. A thorough explanation of the HLA MOM including Object Model Template descriptions is provided in "High Level Architecture Management Object Model" located at the DMSO web-site.

RTI 1.0 implements the Manager::Federate object class which the RTI is responsible for publishing, updating, and receiving reflected attribute values. This class provides the federation with information about the federates identity, time settings, RTI version, and internal queue sizes. For a description of the attributes implemented, see Table 2-3: Manager::Federate Attribute Definitions.

Table 2-3: Manager::Federate Attribute Definitions

Attribute	Definition
FederateHost	The string representation of the hostname the federate is executing on.
FederateHandle	The string representation of an integer that is the handle assigned to the federate by the fedex.
FederateState	The string representation of the integer corresponding to the value of the *RTI::FederateStateType* enumeration appropriate for the federate.

Table 2-3: Manager::Federate Attribute Definitions

Attribute	Definition
FederateName	The string representation of the name specified by the federate at join time - it is for descriptive purposes only.
RTIversion	The string representation of the software version of the RTI library. The initial 1.0 release is version **1.0.1**.
TimeManagerState	The string representation of the integer corresponding to the value of the *RTI::TimeManagerStateType* enumeration appropriate for the federate. This value indicates what type of time-advancement service (if any) is currently in effect for the federate.
FederateLookahead	The string representation of a double that is the value of the federate's lookahead.
FederateTime	The string representation of a double that is the value of the federate's local time.
TimeConstrained	The character representation of an integer that specifies whether the federate is constrained or unconstrained, where 0 is False and 1 is True.
TimeRegulating	The character representation of an integer that specifies whether the federate is regulating or not regulating, where 0 is False and 1 is True.
FIFOlength	The string representation of an integer that specifies the number of elements in the First In First Out (FIFO) queue. The FIFO queue contains messages sent with RECEIVE order.
TSOlength	The string representation of an integer that specifies the number of elements in the Time Stamp Ordered (TSO) queue. The TSO queue contains messages sent with RECEIVE order.
DequeueFIFOasync	The string representation of the boolean value indicating whether or not the federate is asynchronously processing receive-order messages. (See the *RTIambassador::dequeueFIFOasynchronously* service for further description.)
TotalObjectCount	The string representation of an integer that specifies the total number of objects known to the federate (either discovered or registered and is reduced by delete and remove object services).
HoldingTokensObjectCount	The string representation of an integer that specifies the number of objects in a federates database but the federate is not aware of. This occurs when someone resigns and releases ownership of tokens but no one else assumed ownership. This is specific to the 1.0 implementation and may not be generally necessary.
DeletedObjectCount	The string representation of an integer that specifies the number of objects the federate knows about that have been deleted. This is specific to the 1.0 implementation and may not be generally necessary. It is used to ensure that deleted objects are not rediscovered due to traffic latencies. An object will remain in the database for the number of ticks specified in the RID file. This value may need to be adjusted due to federation specific run-time properties.
NumAttributes	The string representation of an integer that acts as an indicator of the number of attribute values stored by the federate.
NumParameters	The string representation of an integer that acts as an indicator of the number of parameter values stored by the federate.

The Manager::Federation class provides information about the federation state and is published by the RTI (one per federation.) For a description of the attributes implemented, see Table 2-4: Manager::Federation Attribute Definitions.

Table 2-4: Manager::Federation Attribute Definitions

Parameter	Definition
FederationName	The string name of the federation.
FederationState	The string representation of the integral value of the RTI::FederationStateType enumeration. This value indicates the pause/resume state of the federation as a whole
FederatesInFederation	The string representation of the integral number of federates joined in the federation execution.
SaveIsScheduled	The string representation of the boolean value indicating whether or not a federation save is currently scheduled.
ScheduledSaveTime	The string representation of the double-precision floating-point number representing the logical time of the scheduled federation save (or positive infinity if no save is scheduled.)
RTIversion	The string representation of the version number of the federation executive (in the initial release of RTI 1.0, this is **1.0.1**.)

The Manager::Federate::Alert interaction allows the RTI to inform the federation when an exceptional condition occurs in a federate. For a description of the parameters of this interaction, see Table 2-5: Manager::Federate::Alert Parameter Definitions.

Table 2-5: Manager::Federate::Alert Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
AlertSeverity	The string representation of the integral value of the LogType enumeration. Possible values are RTI_EXCEPTION=0, RTI_INTERNAL_ERROR, RTI_FEDERATE_ERROR, RTI_WARNING, RTI_DIAGNOSTIC.
AlertText	The string representation of the reason of the alert.
AlertID	The string representation of the serial number for an exception.

The Manager::Federate::ServiceLog interaction allows detailed tracing of *RTIambassador* and *FederateAmbassador* method invocations. The generation of such can be toggled using the Manager::Action::Control interaction. For a description of the attributes implemented, see Table 2-6: Manager::Federate::ServiceLog Parameter Definitions.

Table 2-6: Manager::Federate::ServiceLog Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ServiceName	The string method name of the service call generating the interaction.
ServiceInitiator	The string representing the initiator of the service call (**FED** for *RTIambassador* methods or **RTI** for *FederateAmbassador* methods.)

The Manager::Federate::ServiceLog::ServiceLogArguments interaction allows detailed tracing of *RTIambassador* and *FederateAmbassador* method invocations including the arguments provided in each call. The generation of such can be toggled using the Manager::Action::Control interaction. For a description of the attributes implemented, see Table 2-7: Manager::Federate::ServiceLog::ServiceLogArguments Parameter Definitions.

Table 2-7: Manager::Federate::ServiceLog::ServiceLogArguments Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ServiceName	The string method name of the service call generating the interaction.
ServiceInitiator	The string representing the initiator of the service call (**FED** for *RTIambassador* methods or **RTI** for *FederateAmbassador* methods.)
Handle1	Meaning is dependent on service invoked. Parameter is represented as a string.
Handle2	Meaning is dependent on service invoked. Parameter is represented as a string.
HandleSet	Meaning is dependent on service invoked. Parameter is represented as a string.
ObjectIDorCount	Meaning is dependent on service invoked. Parameter is represented as a string.
TagOrLabelOrName	Meaning is dependent on service invoked. Parameter is represented as a string.
Time	The string representation of the time provided to the service invoked.
Enumeration	Meaning is dependent on service invoked. Parameter is represented as a string.
Boolean	Meaning is dependent on service invoked. Parameter is represented as a string.

The Manager::Federate::ObjectInformation interaction is sent by the RTI in response to a Manager::Federate::Action::RequestObjectInformation interaction sent by a federate. It reports information about the internal state the RTI maintains for the object. For a description of the parameters implemented, see Table 2-8: Manager::Federate::ObjectInformation Parameter Definitions.

Table 2-8: Manager::Federate::ObjectInformation Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ObjectID	The string representation of the ObjectID that this interaction id reporting information on.
LockedAttributes	The string representation of the attributes that are owned by a federate.
RegisteredClass	The string representation of the class that was registered by the registering federate.
RepresentedClass	The string representation of the class that was discovered by the fromFederate.

The Manager::Federate::PublishingClass interaction is sent by the RTI in response to a Manager::Federate::Action::RequestPublicationTree interaction sent by a federate. It reports the current

publication state for a federate. For a description of the parameters implemented, see Table 2-9: Manager::Federate::PublishingClass Parameter Definitions.

Table 2-9: Manager::Federate::PublishingClass Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ObjectClass	The string representation of the object class and attributes published for one object class. The format of the string is "ClassHandle".
InteractionClass	The string representation of the interaction class handle. The format of the string is "ClassHandle:attributeHandle,attributeHandle,...,attributeHandle".

The Manager::Federate::SubscribingClass interaction is sent by the RTI in response to a Manager::Federate::Action::RequestSubscriptionTree interaction sent by a federate. It reports the current subscription state for a federate. For a description of the parameters implemented, see Table 2-10: Manager::Federate::SubscribingClass Parameter Definitions.

Table 2-10: Manager::Federate::SubscribingClass Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ObjectClass	The string representation of the object class and attributes published for one object class. The format of the string is "ClassHandle".
InteractionClass	The string representation of the interaction class handle. The format of the string is "ClassHandle:attributeHandle,attributeHandle,...,attributeHandle".

The Manager::Federate::Action interaction is used to perform an action on a remote federate or a federate's local RTI component. For a description of the parameters implemented, see Table 2-11: Manager::Federate:Action Parameter Definitions.

Table 2-11: Manager::Federate:Action Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ToFederate	The string representation of the receiving federate's handle.

The Manager::Federate::Action::RequestPublicationTree interaction is used to request that the RTI provide the federate with the current publications of a federate. For a description of the parameters implemented, see Table 2-12: Manager::Federate::Action::RequestPublicationTree Parameter Definitions.

Table 2-12: Manager::Federate::Action::RequestPublicationTree Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ToFederate	The string representation of the receiving federate's handle.

The Manager::Federate::Action::RequestSubscriptionTree interaction is used to request that the RTI provide the federate with the current subscriptions of a federate. For a description of the parameters implemented, see Table 2-13: Manager::Federate::Action::RequestSubscriptionTree Parameter Definitions.

Table 2-13: Manager::Federate::Action::RequestSubscriptionTree Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ToFederate	The string representation of the receiving federate's handle.

The Manager::Federate::SetTiming interaction allows modification of a federate's periodic rate that the RTI will automatically update the Manager::Federate attributes related to federate identity, time, and objects respectively. The default value for the periodic rate is positive infinity - this means only one update occurs at startup by default. This value can also be specified by the MOM_TIME_RESOLUTION value in the RID file. For a description of the parameters implemented in this interaction, see Table 2-14: Manager::Federate:SetTiming Parameter Definitions.

Table 2-14: Manager::Federate:SetTiming Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ToFederate	The string representation of the federate's handle that the interaction is intended to modify.
FedReportPeriod	The string representation of the integer that is the number of seconds between MOM updates of federate attributes (Hostname, federate name, fedex name, federate handle, and RTI version).
TimeReportPeriod	The string representation of the integer that is the number of seconds between MOM updates of federate time attributes(lookahead, federate time, constrained, regulating, TSO length, and FIFOlength)
ObjectReportPeriod	The string representation of the integer that is the number of seconds between MOM updates of federate object attributes (TotalObjectCount, HoldingTokensObjectCount, DeletedObjectCount, NumAttributes, NumParameters, NumBytesInAttributes, and NumBytesInParamaters)

The Manager::Federate::Action::RequestObjectInformation interaction causes the RTI to send a Manager::Federate::ObjectInformation interaction informing a federate of the state it is maintianing for a specific object. For a description of the parameters implemented in this interaction, see Table 2-15: Manager::Federate::Action::RequestObjectInformation Parameter Definitions.

Table 2-15: Manager::Federate::Action::RequestObjectInformation Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ToFederate	The string representation of the federate's handle that the interaction is intended to modify.
ObjectID	The string representation of the ObjectID that information is being requested for.

The Manager::Federate::Action::ModifyAttributeState interaction allows federates to modify the ownership token status of an attribute-instance. This should not be attempted by someone unfamiliar with RTI internals. For a description of the parameters implemented in this interaction, see Table 2-16: Manager::Federate::Action::ModifyAttributeState Parameter Definitions.

Table 2-16: Manager::Federate::Action::ModifyAttributeState Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ToFederate	The string representation of the federate's handle that the interaction is intended to modify.
ObjectID	The string representation of the object whose attribute token status is to be modified.
AttributeID	The string representation of the attribute whose instance's token status is to be modified.
TokenState	The string representation of the integral value of the RTI::TokenState enumeration to set the attribute-instance to.

The Manager::Federate::Action::RemoteServiceInvocation::DoResignFederationExecution interaction executes the equivalent of the *RTIambassador::resignFederationExecution* service on the target federate. For a description of the parameters implemented in this interaction, see Table 2-17: Manager::Federate::Action::RemoteServiceInvocation::DoResignFederationExecution Parameter Definitions.

Table 2-17: Manager::Federate::Action::RemoteServiceInvocation::DoResignFederationExecution Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ToFederate	The string representation of the federate's handle that the interaction is intended to modify.
ResignAction	The string representation of the integral value of the RTI::ResignAction enumeration to use as an argument to the *resignFederationExecution*. service.

The Manager::Federate::Action::RemoteServiceInvocation::DoDeleteObject interaction executes the equivalent of the *RTIambassador::deleteObject* on the target federate. For a description of the parameters implemented in this interaction, see Table 2-18: Manager::Federate::Action::RemoteServiceInvocation::DoDeleteObject Parameter Definitions.

Table 2-18: Manager::Federate::Action::RemoteServiceInvocation::DoDeleteObject Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ToFederate	The string representation of the federate's handle that the interaction is intended to modify.
ObjectID	The string representation of the object ID to use as an argument to the

Parameter	Definition
	deleteObject service.
Time	The string representation of the federatin time to use as an argument to the *deleteObject* service.
Tag	The string to use as an argument to the *deleteObject* service.

The Manager::Federate::Action::RemoteServiceInvocation::DoSetLookahead interaction executes the equivalent of the *RTIambassador::setLookahead* method on the target federate. For a description of the parameters implemented in this interaction, see Table 2-19: Manager::Federate::Action::RemoteServiceInvocation::DoSetLookahead Parameter Definitions.

Table 2-19: Manager::Federate::Action::RemoteServiceInvocation::DoSetLookahead Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ToFederate	The string representation of the federate's handle that the interaction is intended to modify.
Lookahead	The string representation of a double that is the value the federate's lookahead will be set to. The lookahead value is specified as a double.

The Manager::Federate::Action::RemoteServiceInvocation::DoSetTimeConstrained interaction executes the equivalent of the *RTIambassador::setTimeConstrained* method on the target federate. For a description of the parameters implemented in this interaction, see Table 2-20: Manager::Federate::Action::RemoteServiceInvocation::DoSetTimeConstrained Parameter Definitions.

Table 2-20: Manager::Federate::Action::RemoteServiceInvocation::DoSetTimeConstrained Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ToFederate	The string representation of the federate's handle that the interaction is intended to modify.
State	The string representation of an integer (True=0, False=1) that toggles whether the federate is constrained or not constrained. The specified value gets passed to an invocation of setTimeConstrained().

The Manager::Federate::Action::RemoteServiceInvocation::DoTurnRegulationOn interaction executes the equivalent of the *RTIambassador::turnRegulationOn* method on the target federate. For a description of the parameters implemented in this interaction, see Table 2-21: Manager::Federate::Action::RemoteServiceInvocation::DoTurnRegulationOn Parameter Definitions.

Table 2-21: Manager::Federate::Action::RemoteServiceInvocation::DoTurnRegulationOn Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.

Parameter	Definition
ToFederate	The string representation of the federate's handle that the interaction is intended to modify.

The Manager::Federate::Action::RemoteServiceInvocation::DoTurnRegulationOff interaction executes the equivalent of the *RTIambassador::turnRegulationOff* method on the target federate. For a description of the parameters implemented in this interaction, see Table 2-22: Manager::Federate::Action::RemoteServiceInvocation::DoTurnRegulationOff Parameter Definitions.

Table 2-22: Manager::Federate::Action::RemoteServiceInvocation::DoTurnRegulationOff Parameter Definitions

Parameter	Definition
FromFederate	The string representation of the initiating federate's handle.
ToFederate	The string representation of the federate's handle that the interaction is intended to modify.

3. HLA Service to C++ Mapping

This section describes the mapping from the HLA 1.0 Interface Specification to C++. The focus of this section is to describe the services that are implemented, the methods that should be used together and the memory allocation rules for key methods. Note: Only the services that are implemented in F.0 are described in this section.

3.1 Federation Management

Table 3-23: Federation Management Services

Section	Service Title	Service Implemented
2.1	Create Federation Execution	Yes
2.2	Destroy Federation Execution	Yes
2.3	Join Federation Execution	Yes
2.4	Resign Federation Execution	Yes
2.5	Request Pause	Yes
2.6	Initiate Pause †	Yes
2.7	Pause Achieved	Yes
2.8	Request Resume	Yes
2.9	Initiate Resume †	Yes
2.10	Resume Achieved	Yes
2.11	Request Federation Save	Yes
2.12	Initiate Federate Save †	Yes
2.13	Federate Save Begun	Yes
2.14	Federate Save Achieved	Yes
2.15	Request Restore	Yes
2.16	Initiate Restore †	Yes
2.17	Restore Achieved	Yes

3.1.1 Create Federation Execution

NAME

createFederationExecution - create a named federation execution and register it with the RTI executive

HLA INTERFACE SPECIFICATION SERVICE

2.1 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::createFederationExecution (
  const RTI::FederationExecutionName executionName
)
 throw (
  RTI::FederationExecutionAlreadyExists,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

executionName

string specifying the name of the federation execution to create. The caller is responsible for freeing the memory used by this string and may do so at any time after the completion of the call.

DESCRIPTION

This method first queries the RTI executive to determine if a federation executive already exists for the given federation. If not, it attempts to fork a subprocess to run the federation execution executable located at *$RTI_HOME/bin/fedex.sh*. Upon successful initialization, the federation executive will inform the RTI executive of its existence; only at this point may federates begin joining the execution.

RETURN VALUES

A non-exceptional exit from this method indicates that the RTI executive has approved the creation of the named federation execution and that the federation execution subprocess has been forked.

It is important to note that a non-exceptional return does not guarantee that the federation execution has been successfully created; errors that occur in the initialization of the federation execution subprocess (e.g. bad path to the federation execution executable) are not detected by the *RTIambassador::createFederationExecution* method. Output from the federation executive is logged to a file **Xterm.#####** (where ##### is a PID) in the current directory; this log should be consulted when attempting to diagnose problems with federation executive initialization.

Even when the federation executive initialization is successful, there is necessarily a

(non-negligible) period of time between the return of the *RTIambassador::createFederationExecution* call and the time that the federation execution is initialized, registered, and ready to accept joining federates. As a result, calls to *RTIambassador::joinFederationExecution* immediately following calls to *RTIambassador::createFederationExecution* will fail; applications should have a time delay between these two calls and/or be prepared to call *RTIambassador::joinFederationExecution* multiple times to ensure that the federation executive has had time to be initialized.

WINDOWS NT NOTES

On Windows NT, the path of the executable that is forked to start the federation executive is *%RTI_HOME%.exe*. Currently, output from the federation executive is not logged on Windows NT.

EXCEPTIONS

RTI::FederationExecutionAlreadyExists - A federation executive for the given federation has already been registered with the RTI executive. Federation executives will unregister themselves with the RTI executive upon termination; however, sometimes an abnormal termination of a federation executive (e.g. a "kill -9") will result in a defunct federation executive still being registered. If this occurs, it is necessary to unregister the federation executive manually via the RTI executive console interface.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::joinFederationExecution,
RTIambassador::destroyFederationExecution

3.1.2 Destroy Federation Execution

NAME

destroyFederationExecution - unregister a named federation execution with the RTI executive and shut down the federation executive

HLA INTERFACE SPECIFICATION SERVICE

2.2 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::destroyFederationExecution (
  const RTI::FederationExecutionName federationName
)
 throw (
  RTI::FederatesCurrentlyJoined,
  RTI::FederationExecutionDoesNotExist,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

federationName

string specifying the name of the federation execution to destroy. The caller is responsible for freeing the memory used by this string and may do so at any time after the completion of the call.

DESCRIPTION

This method queries the RTI executive for the location of the federation executive for the given federation. If one exists, it is asked to destroy itself. If the prerequisites for federation executive destruction are met (i.e. there are no federates joined to the federation execution), the federation executive notifies the RTI executive of its intention to shut down and exits.

There are no restrictions on whom may destroy the federation execution; a federate need not be the creator of the federation executive, or even have been a member of the federation execution.

RETURN VALUES

A non-exceptional return indicates that the federation execution has been successfully destroyed.

EXCEPTIONS

RTI::FederatesCurrentlyJoined - There are still federates joined in the federation execution. All federates must have resigned (or been manually removed via the federation executive console) from the federation execution before the it can be destroyed. Note that the federation executive automatically removes non-existent federates from the federation (even if they fail to resign properly), so it shouldn't be necessary to manually remove federates under normal circumstances.

RTI::FederationExecutionDoesNotExist - The RTI does not have a federation executive registered for the given federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTIambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::createFederationExecution

3.1.3 Join Federation Execution

NAME

joinFederationExecution - request permission to participate in a named federation execution from the federation executive and associate the RTI ambassador with the federation execution

HLA INTERFACE SPECIFICATION SERVICE

2.3 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

RTI::FederateHandle
joinFederationExecution (
  const RTI::FederateName yourName
  const RTI::FederationExecutionName executionName
  RTI::FederateAmbassadorPtr federateAmbassadorReference
)
 throw (
  RTI::FederateAlreadyExecutionMember,
  RTI::FederationExecutionDoesNotExist,
  RTI::CouldNotOpenFED,
  RTI::ErrorReadingFED,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

yourName

string indicating the symbolic name by which the federate will be known to the federation.

executionName

string indicating the name of the federation execution to join.

The caller is responsible for freeing storage space used by these strings and may do so at any time after the completion of the call.

federateAmbassadorRef

pointer to an instance of an application-defined subclass of *RTI::FederateAmbassador* on which RTI-initiated callbacks will be invoked. The caller is responsible for freeing the storage associated with this object, but it should not do so until the object is no longer needed by the RTI ambassador (i.e. *RTIambassador::resignFederationExecution* has been called.)

DESCRIPTION

This method queries the RTI executive for the location of the federation executive responsible for the given named federation execution. It then issues a request to the federation executive to join the federation execution. Lastly, RTI ambassador

internals are initialized using the Federation Execution Data (FED) file whose path is given by *$RTI_CONFIG/[federation name].fed*.

Upon successful completion, the RTI ambassador is associated with a particular federation execution and will notify the federate of changes in federation state through the invocation of *federateAmbassadorRef* callbacks. Keep in mind that no data will be presented to the federate ambassador until the federate has declared interest via the appropriate declaration management services.

Also upon successful completion, the Management Object Model (MOM) Manager has published an object of class *Federate* representing the local federate and sent out an initial attribute update for this object.

RETURN VALUES

The federate handle returned by this method is a numeric value that the RTI has associated with the federate (precisely, it represents the federate's offset in a "federate vector" that is used internally by the RTI.) In the 1.0 RTI, this value is probably of very little interest to the application.

WINDOWS NT NOTES

On Windows NT, the path of the FED file is given by *%RTI_HOME%federation name].fed*.

EXCEPTIONS

RTI::FederateAlreadyExecutionMember - The RTI ambassador is already associated with a federation execution. An RTI ambassador may only be associated with one federation execution at a given time (although the same RTI ambassador may be associated with different federation executions at different times and different RTI ambassadors may be associated with different federation executions at the same time.)

RTI::FederationExecutionDoesNotExist - There was no federation executive registered for the given named federation execution.

RTI::CouldNotOpenFED - The FED file could not be found at *$RTI_CONFIG/[federation name].fed*.

RTI::ErrorReadingFED - The FED file was not in the correct format. This can occur if one of the classes or interactions used by the MOM manager is missing or incorrect; see the example FED files in the RTI distribution for the definitions of MOM data types.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::createFederationExecution,
RTIambassador::resignFederationExecution RTIambassador::publishObjectClass,
RTIambassador::publishInteractionClass
RTIambassador::subscribeObjectClassAttribute,
RTIambassador::subscribeInteractionClass RTIambassador::turnRegulationOn,
RTIambassador::setTimeConstrained

3.1.4 Resign Federation Execution

NAME

resignFederationExecution - resolve ownership of attributes and notify the federation executive that the federate no longer wishes to participate in the federation execution

HLA INTERFACE SPECIFICATION SERVICE

2.4 - *Federation Management* (federate initiated)

SYNOPSIS

enum RTI::ResignAction *RELEASE_ATTRIBUTES* = 1, *DELETE_OBJECTS*, *DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES*, *NO_ACTION* ;

```
void
RTIambassador::resignFederationExecution (
  RTI::ResignAction theAction
)
 throw (
  RTI::FederateOwnsAttributes,
  RTI::FederateNotExecutionMember,
  RTI::InvalidResignAction,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theAction

enumerated value indicating the desired policy for relinquishment of federate-owned attributes.

DESCRIPTION

This method informs the federation executive that the federate no longer wishes to participate in the federation execution. Before doing so, it is necessary to resolve ownership of any object-attributes owned by the federate. The four resolution policies defined are:

RELEASE_ATTRIBUTES

federate releases control of any owned attributes (including *privilege-to-delete* attributes) before resigning. This is similar to doing an unconditional divestiture of every attribute owned by the federate. Any ownership tokens that aren't assumed by another federate become "orphaned". Orphaned tokens continue to exist in the federation (specifically, they are tracked by the RTI internally by another federate process or by the federation executive) and are eligible for acquisition by any interested federate (however, no notification of the existance of such attributes is provided other than the initial *RTI::requestAttributeOwnershipAssumption*.)

DELETE_OBJECTS

resigning federate deletes all objects for which it holds the privilege to delete (i.e. owns the *privilegeToDelete* attribute that is implicitly defined for every object.) The effect of this option is the same as if the federate had explicitly

called *RTIambassador::deleteObject* for every object for which it holds the *privilegeToDelte* token. If the federate owns attributes of objects for which it does not hold the delete privilege, these attributes become "zombies" (see *NO ACTION* below.)

DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES
resigning federate first deletes any objects for which the federate holds the delete privilege (see *DELETE_OBJECTS* above), then releases ownership of any remaining owned attributes (see *RELEASE_ATTRIBUTES* above.) This is probably the best option to use in most situations.

NO_ACTION
action is taken upon resignation; all attributes and objects owned by the federate will become "zombies", i.e. technically still in existance in the federation but immutable, non-discoverable and not eligible for acquisition by other federates.

The *RTIambassador::resignFederationExecution* method will not return until the ownership of all federate-owned attributes has been resolved as prescribed by the resign action and the connection between the federate and the federation execution has been terminated. At this time, the internal state of the RTI ambassador will have been reset, allowing it to be associated with another federation execution through a subsequent invocation of *RTIambassador::joinFederationExecution*. The federate ambassador associated with the federation execution is no longer needed at this point and may be disposed of at the federate's leisure.

Any messages queued for delivery to the federate at the time of resignation will be lost.

RETURN VALUES

A non-exceptional return indicates that the resignation was successful, as described in the previous section.

EXCEPTIONS

RTI::FederateOwnsAttributes - Not thrown in 1.0.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::InvalidResignAction - The parameter specifying the ownership resolution policy was not a recognized value.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

RTI::UnimplementedService - An invalid variation of the method was invoked.

SEE ALSO

RTIambassador::joinFederationExecution,
RTIambassador::requestAttributeOwnershipDivestiture,

FederateAmbassador::requestAttributeOwnershipAssumption, RTIambassador::deleteObject , RTIambassador::destroyFederationExecution

3.1.5 Request Pause

NAME

requestPause - request that all federates in the federation suspend execution (as defined by a "pause label") as soon as possible

HLA INTERFACE SPECIFICATION SERVICE

2.5 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::requestPause (
  const RTI::PauseLabel label
)
 throw (
  RTI::FederationAlreadyPaused,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

label

null-terminated string that is passed to the corresponding invocations of *FederateAmbassador::initiatePause*. This parameter is not interpreted by the RTI itself, but is provided as a means for the requesting federate to specify a textual description of the reason for the pause request or any other information relevant in the context of the federation. The federate is responsible for freeing the storage associated with this string and may do so at its leisure upon completion of the call.

DESCRIPTION

This method notifies all remote federates of the federate's desire to suspend federation execution, resulting in invocations of the *FederateAmbassador::initiatePause* method for each federate in the federation execution. Upon receipt of such a request, federates are expected to suspend their execution as soon as possible and notify the RTI via the *RTIambassador::pauseAchieved* method when this has been accomplished.

Note that the RTI does not attach any meaning to the notion of "pause"; federation developers may define different types of pauses associated with different labels in a way that makes sense in the context of a given federation. In particular, pausing a federate does not preclude the sending and recipt of updates and interactions or the utilization of any other RTI services by that federate during the pause period.

The RTI does not currently define a mechanism by which a federate is automatically notified when a requested pause has been achieved; currently the best way to do this is to periodically call *RTIambassador::initiatePause* until the *RTI::FederationAlreadyPaused* exception is thrown.

RETURN VALUES

A non-exceptional return indicates that federate has successfully communicated its desire to suspend federation execution and that the federation execution is not already paused.

EXCEPTIONS

RTI::FederationAlreadyPaused - An attempt was made to suspend execution of an already-paused federation.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::pauseAchieved, FederateAmbassador::initiatePause, RTIambassador::requestResume, RTIambassador::initiateResume

3.1.6 Initiate Pause +

NAME

initiatePause - instructs the federate to suspend execution (as defined by a "pause label") as soon as possible

HLA INTERFACE SPECIFICATION SERVICE

2.6 - *Federation Management* (RTI initiated)

SYNOPSIS

```
#include <RTI.hh>

virtual
void
FederateAmbassador::initiatePause (
 const RTI::PauseLabel label
)
 throw (
  RTI::FederateAlreadyPaused,
  RTI::FederateInternalError
 )
```

ARGUMENTS

label

null-terminated string value that was supplied as the argument to the *RTIambassador::requestPause* method that requested the pause. This parameter is not interpreted by the RTI itself, but is provided as a means for the requesting federate to specify a textual description of the reason for the pause request or any other information relevant in the context of the federation. The RTI retains ownership of the storage associated with this string, so the federate must make a copy if it wishes to retain its value after the completion of the call.

DESCRIPTION

This callback is invoked in response to a pause request (*RTIambassador::requestPause*) made by a remote federate. Note that the RTI does not attach any meaning to the notion of "pause"; federation developers may define different types of pauses associated with different labels in a way that makes sense in the context of a given federation. In particular, pausing a federate does not preclude the sending and recipt of updates and interactions or the utilization of any other RTI services by that federate during the pause period.

A pause-initiation request supercedes any previous requests; the pause label given as an argument to *RTIambassador::pauseAchieved* should be the pause label associated with the most recent pause-initiation request.

RETURN VALUES

A non-exceptional return indicates that the federate is not already paused and will attempt to suspend execution in accordance with the pause request as soon as possible.

EXCEPTIONS

RTI::FederateAlreadyPaused - The federate has already suspended its execution and

notified the RTI of such via the *RTIambassador::pauseAchieved* method.

RTI::FederateInternalError - An error internal to the federate has occurred; this exception will result in an entry being made to the federate's RTI log.

SEE ALSO

RTIambassador::pauseAchieved, RTIambassador::requestPause, FederateAmbassador::initiateResume

3.1.7 Pause Achieved

NAME

pauseAchieved - inform the RTI that the federation has suspended execution as per the most recent pause request

HLA INTERFACE SPECIFICATION SERVICE

2.7 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::pauseAchieved (
  const RTI::PauseLabel label
)
 throw (
  RTI::UnknownLabel,
  RTI::NoPauseRequested,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

label

null-terminated "pause label" associated with the pause request. This should be the same as the argument to the most recent *FederateAmbassador::initiatePause* invocation. The federate is responsible for freeing the memory used by this string and may do so at any time upon the completion of the call.

DESCRIPTION

This method informs the RTI that the federate has successfully achieved a suspension of execution, as prescribed by the pause label argument to the most recent *FederateAmbassador::initiatePause* invocation. The federate should remain suspended in accordance with the terms of the pause label until instructed to resume via a *FederateAmbassador::initiateResume* notification.

RETURN VALUES

A non-exceptional return indicates that the RTI has been notified of the federate's successful suspension of execution.

EXCEPTIONS

RTI::UnknownLabel - The label provided does not match the label associated with the most recent outstanding pause request.

RTI::NoPauseRequested - There is not an outstanding pause request.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::initiatePause, RTIambassador::requestPause, RTIambassador::requestResume, FederateAmbassador::initiateResume

3.1.8 Request Resume

NAME

requestResume - request that a paused federation resume execution as soon as possible

HLA INTERFACE SPECIFICATION SERVICE

2.8 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::requestResume ( )
 throw (
  RTI::FederationNotPaused,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError,
  RTI::SaveInProgress,
  RTI::RestoreInProgress
 )
```

DESCRIPTION

This service instructs the federates in the federation execution to resume execution as soon as possible after the successful completion of a federation pause. The *FederateAmbassador::initiateResume* method of paused remote federates will be invoked to notify them of the continuance of federation execution.

There is currently no easy way for a federate to determine when all federates have been resumed execution. The Management Object Model currently provides the only facility for doing this.

The federate requested the continuance of federation execution need not be the same federate that requested the pause.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully communicated its desire to resume federation execution.

EXCEPTIONS

RTI::FederationNotPaused - The federation execution is not currently in a suspended state.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::resumeAchieved, FederateAmbassador::initiateResume, RTIambassador::requestPause

3.1.9 Initiate Resume +

NAME

initiateResume - instructs a paused federate to resume execution as soon as possible

HLA INTERFACE SPECIFICATION SERVICE

2.9 - *Federation Management* (RTI initiated)

SYNOPSIS

```
#include <RTI.hh>

virtual
void
FederateAmbassador::initiateResume ( )
 throw (
   RTI::FederateNotPaused,
   RTI::FederateInternalError
 )
```

DESCRIPTION

This callback is invoked in response to another federate's request to resume federation execution (via the *RTIambassador::requestResume* method.) The federate should resume execution as soon as possible and notify the RTI of such using the *RTIambassador::resumeAchieved* service.

RETURN VALUES

A non-exceptional return indicates that the federate is currently paused and will resume execution as soon as possible.

EXCEPTIONS

RTI::FederateAlreadyPaused - The federate is not currently in a state of suspended execution.

RTI::FederateInternalError - An error internal to the federate has occurred; this exception will result in an entry being made to the federate's RTI log.

SEE ALSO

RTIambassador::resumeAchieved, RTIambassador::requestResume, FederateAmbassador::initiatePause

3.1.10 Resume Achieved

NAME

resumeAchieved - notify the RTI that the federate has resumed execution as per an initiate resume request

HLA INTERFACE SPECIFICATION SERVICE

2.10 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::requestResume ( )
 throw (
  RTI::FederationNotPaused,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

DESCRIPTION

This service method is invoked to inform the RTI that the federate has resumed execution in accordance with an outstanding *FederateAmbassador::initiateResume* request.

RETURN VALUES

A non-exceptional return indicates that the federate has communicated its desire to resume execution.

EXCEPTIONS

RTI::FederationNotPaused - The federate is not currently in a suspended state.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::initiateResume, RTIambassador::requestResume,

FederateAmbassador::initiatePause

3.1.11 Request Federation Save

NAME

requestFederationSave - request that the federation save its state at a specified logical time

HLA INTERFACE SPECIFICATION SERVICE

2.11 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::requestFederationSave (
  const RTI::SaveLabel label
  RTI::FederationTime theTime
)
 throw (
  RTI::FederationTimeAlreadyPassed,
  RTI::InvalidFederationTime,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

A variation initiates the save as soon as possible regardless of the logical times of the federates:

```
void
requestFederationSave (
  const RTI::SaveLabel label
)
 throw (
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

label

null-terminated string that is passed to the resulting invocations of *FederateAmbassador::initiateFederateSave*. This parameter is not interpreted by the RTI itself, but is provided as a means for the requesting federate to specify a textual description of the reason for the restore request or any other information meaningful in the context of the federation. This label is also a component of the filenames in which the RTI's internal state is expected to be saved, allowing for differentiation among multiple saved states. The federate is responsible for freeing the storage associated with this string and may do so at its leisure upon completion of the call.

theTime
logical time that the federation save is to take place at (omission of this argument implies that the save should take place as soon as possible.)

DESCRIPTION

This service allows the federate to initiate the federation save process. It will result in invocations of *FederateAmbassador::initiateFederateSave* being scheduled for each remote federate. If a logical time for the save is not specified, such a callback will immediately be queued for delivery (note that this will usually result in different federates being saved at 0000000different logical times.)

RETURN VALUES

A non-exceptional return indicates that the federation save has been initiated.

EXCEPTIONS

RTI::FederationTimeAlreadyPassed - The requested save time is less than the current effective federate logical time (i.e. the federate's logical time plus lookahead.)

RTI::InvalidFederationTime - The requested save time is invalid because it is less than the effective logical time of one or more federates currently joined in the federation execution.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::initiateFederateSave, RTIambassador::federateSaveBegun, FederateAmbassador::requestRestore, RTIambassador::federateSaveAchieved, RTIambassador::federateSaveNotAchieved, FederateAmbassador::timeAdvanceGrant, RTIambassador::requestFederateTime

3.1.12 Initiate Federate Save +

NAME

initiateFederateSave - instructs the federate to save its state as of its current logical time

HLA INTERFACE SPECIFICATION SERVICE

2.12 - *Federation Management* (RTI initiated)

SYNOPSIS

```
#include <RTI.hh>

virtual
void
FederateAmbassador::initiateFederateSave (
 const RTI::SaveLabel label
)
 throw (
  RTI::UnableToPerformSave,
  RTI::FederateInternalError
 )

virtual
void
FederateAmbassador::initiateFederateSave (
 const RTI::SaveLabel label
 RTI::FederationTime theTime
)
 throw (
  RTI::InvalidFederationTime,
  RTI::UnableToPerformSave,
  RTI::FederateInternalError
 )
```

ARGUMENTS

label

null-terminated string that was passed to invocation of the *RTIambassador::requestFederationSave* service that requested the save. This parameter is not interpreted by the RTI itself, but is provided as a means for the requesting federate to specify a textual description of the reason for the save request or any other information meaningful in the context of the federation. The federate must make a copy of this parameter if it wishes to retain its value after the completion of the call.

theTime

federation save time as specified to the invocation of the *RTIambassador::requestFederationSave* service that requested the save. Note that this parameter is not really necessary as the RTI schedules the federate save initiation at the appropriate time, i.e. such that the federate should always begin its save as soon as possible after the receipt of this callback.

DESCRIPTION

This callback instructs the federate to begin saving its state as soon as possible. The federate should save its state as of the current federate logical time (*RTIambassador::requestFederateTime*), which should be the same as the *theTime* parameter (if present.) The RTI will not initiate the federate save until it has determined that it is "safe" for the federate to advance to the save time, i.e. the conditions necessary for a *FederateAmbassador::timeAdvanceGrant* to the save time have been met.

Upon the receipt of such a callback, the federate's logical time will not advance until the federate has achieved the save or indicated that the save could not be achieved. In particular, no time-stamp-ordered events with a time greater than the save time will be delivered to the federate, and no *FederateAmbassador::timeAdvanceGrant* will be made.

The federate should notify the RTI using the *RTIambassador::federateSaveBegun* service when it has begun the save process.

RETURN VALUES

A non-exceptional return value indicates that the federate has acknowledged the save initiation request and will begin saving its state as soon as possible, notifying the RTI of such via the *RTIambassador::federateSaveBegun* service.

EXCEPTIONS

RTI::InvalidFederationTime - The specified time is less than the federate's current logical time or greater than the time that the federation mostly recently requested to advance to.

RTI::UnableToPerformSave - The federate is unable to perform a save at the current time.

RTI::FederateInternalError - An error internal to the federate has occurred; this exception will result in an entry being made to the federate's RTI log.

SEE ALSO

RTIambassador::requestFederationSave, RTIambassador::federateSaveBegun, FederateAmbassador::requestRestore, RTIambassador::federateSaveAchieved, RTIambassador::federateSaveNotAchieved, FederateAmbassador::timeAdvanceGrant, RTIambassador::requestFederateTime

3.1.13 Federate Save Begun

NAME

federateSaveBegun - notify the RTI that the federate has begun saving its internal state as per an *initiateFederateSave* request

HLA INTERFACE SPECIFICATION SERVICE

2.13 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::federateSaveBegun ( )
 throw (
  RTI::SaveNotInitiated,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

A variation is unimplemented:

```
void
RTIambassador::federateSaveBegun (
  RTI::FederationTime theTime
)
 throw (
  RTI::SaveNotInitiated,
  RTI::InvalidFederationTime,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError,
  RTI::UnimplementedService
 )
```

ARGUMENTS

theTime (unused)

logical time of the federate as of the save. This parameter is unnecessary, as the federate save necessarily must begin at the same logical time as the *FederateAmbassador::initiateFederateSave* request.

DESCRIPTION

The federate utilizes this service to inform the RTI that it has begun saving its state in compliance with an outstanding *FederateAmbassador::initiateFederateSave* request. The federate should do this as soon as possible after the initiation of the save and will be unable to advance in time (and therefore receive time-stamp-ordered events) or utilize any other service that would change the internal state of the RTI until the completion of the save.

The federate should notify the RTI using the

RTIambassador::federateSaveAchieved or *RTIambassador::federateSaveNotAchieved* services when the save has been completed. At this time the RTI will wait for all other federates to complete their saves and then save its internal state.

RETURN VALUES

A non-exceptional return indicates that the RTI acknowledges the beginning of the federate save and the federate may proceed to save its state.

EXCEPTIONS

RTI::SaveNotInitiated - There is no currently outstanding request for a federate save.

RTI::InvalidFederationTime - The federation time argument provided does not match the time of the save, i.e. the federate's current logical time. (Not thrown in RTI 1.0.)

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in the RTI; consult the federate log file for more details.

RTI::UnimplementedService - This service is not implemented in RTI 1.0.

SEE ALSO

FederateAmbassador::initiateFederateSave, *RTIambassador::federateSaveAchieved*, *RTIambassador::federateSaveNotAchieved*

3.1.14 Federate Save Achieved

NAME

federateSaveAchieved, federateSaveNotAchieved - notify the RTI that the federate has completed an attempted federate save

HLA INTERFACE SPECIFICATION SERVICE

2.14 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::federateSaveAchieved ( )
 throw (
  RTI::SaveNotInitiated,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError
 )

void
RTIambassador::federateSaveNotAchieved ( )
 throw (
  RTI::SaveNotInitiated,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError
 )
```

DESCRIPTION

This service should be invoked by the federate when it has completed a federate save begun by an *RTIambassador::federateSaveBegun* service call as per a *FederateAmbassador::initiateFederateSave* request.

This service blocks until all other federates have completed or failed to save their states, then saves the internal state of the RTI in the file **RTIxxx-n.sav** where *xxx* is the save label and *n* is the federate's federate handle. These files will be located in the *$RTI_CONFIG* directory.

If the federate save was not achieved, the RTI will make an entry to the federate's RTI log file and proceed as if the save was successful (i.e. the internal state of the RTI will still be saved.)

Upon return from this method the federate's logical time will continue advancing as prescribed by the time-advance service in effect at the time of the save.

RETURN VALUES

A non-exceptional return indicates that the RTI has successfully saved its internal state and will resume advancement of the federate's logical time.

EXCEPTIONS

RTI::SaveNotInitiated - There is no currently outstanding request for a federate save

or the federate has not indicated the beginning of the save through the *RTIambassador::federateSaveBegun* method.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::RTIinternalError - An internal error has occurred in the RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::requestFederationSave, RTIambassador::federateSaveBegun, RTIambassador::requestRestore, FederateAmbassador::initiateFederateSave

3.1.15 Request Restore

NAME

requestRestore - request that all federates reinitialize themselves based on a labeled save state

HLA INTERFACE SPECIFICATION SERVICE

2.15 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::requestRestore (
 const RTI::SaveLabel label
)
 throw (
  RTI::SpecifiedSaveLabelDoesNotExist,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

label

null-terminated string that is passed to the resulting invocations of *FederateAmbassador::initiateRestore*. This parameter is not interpreted by the RTI itself, but is provided as a means for the requesting federate to specify a textual description of the reason for the restore request or any other information meaningful in the context of the federation. This label is also a component of the filenames in which the RTI's internal state is expected to be saved, so it should match a label that has previously been used as an argument to a save operation. The federate is responsible for freeing the storage associated with this string and may do so at its leisure upon completion of the call.

DESCRIPTION

This service instructs all federates to restore their state to the state saved under the specified label. Unlike the counterpart "save" operation, restoration cannot be scheduled at a specific logical time across all federates; rather, it is initiated immediately upon the receipt of an *RTIambassador::requestRestore* request (this makes sense when you consider that the restored logical times will override the pre-restoration values anyways.) It is often desireable to pause the federation execution before a state restoration.

Upon receipt of such a request by a given federate, the federate's *FederateAmbassador::initiateRestore* callback will be invoked to instruct the federate to immediately begin restoring its state. The RTI does not define a format or provide any facility for federates to save their external state, so it is up to the federate developer to implement the appropriate mechanisms. The RTI is responsible for restoring its internal state and will do so when the federate indicates

(using the *RTIambassador::restoreAchieved* or *RTIambassador::restoreNotAchieved* services) that it has finished restoring its external state.

Only one restoration request may be outstanding at a given (wallclock) time; subsequent invocations of *RTIambassador::requestRestore* will override previous requests.

Note that the RTI attempts to restore its internal state from the file **$RTI_CONFIG/RTIxxx-n.sav** where *xxx* is the restore label and *n* is the federate handle. This means that federates must join the federation execution in a consistent order if the restored internal states are to match up with the same federates.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully communicated its desire to restore federation state from a labelled saved state.

EXCEPTIONS

RTI::SpecifiedSaveLabelDoesNotExist - The file where the RTI expects to find its saved internal state does not exist.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::requestFederationSave, RTIambassador::restoreAchieved, RTIambassador::restoreNotAchieved, FederateAmbassador::initiateRestore

3.1.16 Initiate Restore +

NAME

initiateRestore - instructs the federate to restore its state from a labelled save state

HLA INTERFACE SPECIFICATION SERVICE

2.16 - *Federation Management* (RTI initiated)

SYNOPSIS

```
#include <RTI.hh>

virtual
void
FederateAmbassador::initiateRestore (
  const RTI::SaveLabel label
)
 throw (
   RTI::SpecifiedSaveLabelDoesNotExist,
   RTI::CouldNotRestore,
   RTI::FederateInternalError
 )
```

ARGUMENTS

label

null-terminated string that was given as a parameter to the *RTIambassador::requestRestore* service requesting the restoration. This parameter is not interpreted by the RTI itself, but is provided as a means for the requesting federate to specify a textual description of the reason for the restore request or any other information meaningful in the context of the federation. The federate must make a copy of this parameter if it wishes to retain its value after the completion of the call.

DESCRIPTION

This callback instructs the federate to begin restoring its state immediately. Upon receipt of such a callback, the federate will be in "restore mode" and will be unable to utilize most of the RTI ambassador services (with the notable exception of *RTIambassador::tick*.)

When the federate has restored its state (or failed in the attempt) it should notify the RTI of such using the *RTIambassador::restoreAchieved* or *RTIambassador::restoreNotAchieved* services. At this time the RTI will attempt to reinitialize its internal state from a file whose name is derived from the federate handle and specified save label. See the description of the aforementioned services for more details on this process.

RETURN VALUES

A non-exceptional return value indicates that the federate has acknowledged the request and will begin restoring its state immediately. An exceptional return value will result in an entry being made to the federate's RTI log file; the federate is still expected to proceed with the restoration.

EXCEPTIONS

RTI::SpecifiedSaveLabelDoesNotExist - The specified save label does not

correspond to an existing labelled saved state.

RTI::CouldNotRestore - The federate recognizes the save label but was unable to restore its state for some other reason.

RTI::FederateInternalError - An error internal to the federate has occurred; this exception will result in an entry being made to the federate's RTI log.

SEE ALSO

federateAmbassador::initiateFederateSave, RTIambassador::requestRestore, RTIambassador::restoreAchieved, RTIambassador::restoreNotAchieved

3.1.17 Restore Achieved

NAME

restoreAchieved, restoreNotAchieved - notify the RTI that the federate has completed an attempted federate restoration

HLA INTERFACE SPECIFICATION SERVICE

2.17 - *Federation Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::restoreAchieved ( )
 throw (
  RTI::RestoreNotRequested,
  RTI::RTIcanNotRestore,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError
 )

void
RTIambassador::restoreNotAchieved ( )
 throw (
  RTI::RestoreNotRequested
  RTI::RTIcanNotRestore,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError
 )
```

DESCRIPTION

These services are invoked by the federate when it has completed (or failed to complete) a state restoration as per a *FederateAmbassador::initiateRestore* request.

Both of these services block until all other federates have restored or failed to restore their states, then attempts to restore the internal state of the RTI from the file **RTIxxx-n.sav** where *xxx* is the save label and *n* is the federate's federate handle. These files should be located in the *$RTI_CONFIG* directory.

Note that the *RTIambassador::restoreNotAchieved* method makes an entry to the federate's RTI log file and proceeds as though the restoration had succeeded, i.e. the RTI internal state is still restored.

RETURN VALUES

A non-execptional return indicates that all the federates in the federation have finished restoring their states and that the internal state of the RTI has been restored.

EXCEPTIONS

RTI::RestoreNotRequested - There is no currently outstanding request for a federate restoration.

RTI::RTIcanNotRestore - The RTI internal state save file is missing or corrupt.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::RTIinternalError - An internal error has occurred in the RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::requestRestore, *RTIambassador::requestFederationSave*, *FederateAmbassador::initiateRestore*

3.2 Declaration Management

Table 3-24: Declaration Management Services

Section	Service Title	Service Implemented
3.1	Publish Object Class	Yes
3.2	Publish Interaction Class	Yes
3.3	Subscribe Object Class Attribute	Yes (No regions)
3.4	Subscribe Interaction Class	Yes (No regions)
3.5	Control Updates †	Yes
3.6	Control Interactions †	Yes

3.2.1 Publish Object Class

NAME

publishObjectClass, *unpublishObjectClass* - indicate the intention of the federate to begin (or cease) creating instances and acquiring attributes of a given object class

HLA INTERFACE SPECIFICATION SERVICE

3.1 - *Declaration Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::publishObjectClass (
  RTI::ObjectClassHandle theClass
  const RTI::AttributeHandleSet& attributeList
)
 throw (
  RTI::ObjectClassNotDefined,
  RTI::AttributeNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )

void
RTIambassador::unpublishObjectClass (
  RTI::ObjectClassHandle theClass
)
 throw (
  RTI::ObjectClassNotDefined,
  RTI::FederateOwnsAttributes,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theClass

object class that the federate wishes to begin or cease publication of.

attributeHandleSet

set of attributes of the specified object class that the federate intends to own or update. The caller is responsible for freeing the storage space used by the attribute set and may do so at any time after the completion of the call.

DESCRIPTION

RTIambassador::publishObjectClass indicates to the RTI that the federate is capable of producing updates for the specified attributes of the specified object class. A

federate must be publishing a given object class and attribute to acquire ownership of instances of that attribute, whether through creation of new objects or acquisition of existing attribute-instances through ownership management services. The federate must publish an object class before attempting to register new objects of that class.

The *privilegeToDelete* attribute that is inherently defined for every object class is implicitly published for any object class published by the federate.

Multiple calls to *RTIambassador::publishObjectClass* for the same object class replace the previously published attribute set for the class (henceforth this is referred to as "republishing" the object class.) Publishing an object class does not imply the publication of subclasses; each specific subclass must also be explicitly published.

RTIambassador::unpublishObjectClass removes the specified object class and any of its subclasses from the set of object classes published by the federate. Any valid object class is a valid parameter to the *RTIambassador::unpublishObjectClass* method; the object class need not actually be published by the federate. For example, a federate wishing to unpublish all object classes can do so with one call to *RTIambassador::unpublishObjectClass* with an argument of *ROOT_OBJECT_CLASS_HANDLE*, the handle of the RTI-defined object class from which all federation-defined object classes are implicitly defined.

Note that unpublication only affects the acquisition of new attribute-instances; it does not relieve the federate of update responsibility for any attributes already owned.

Classes derived from the MOM-defined *Manager* object class are handled as a special case by the (un)publication services. All subclasses of *Manager* are implicitly published by the MOM manager and must remain published throughout the lifetime of the federation execution. If the federate attempts to republish a descendent of *Manager* with a different set of attributes, this set must contain all of the attributes pre-defined by the MOM (if it doesn't, the object manager will automatically add these attributes to the new set of published attributes.) *RTIambassador::unpublishObjectClass* will not allow the unpublication of *Manager* descendents. Future versions may allow unpublication of non-MOM-defined attributes; for now this functionality can be achieved by republishing the object class with an attribute set consisting of only the predefined attributes.

Upon publication of a given object class, the federate may receive an *FederateAmbassador::startUpdates* callback instructing it to begin updating a set of attributes of that class. Until the receipt of such a callback, the federate need not send out any updates of the attributes in question to satisfy its publication responsibility. If no other federates have expressed interest in the published attributes, this callback will never be made.

A *FederateAmbassador::stopUpdates* callback negates the effect of the previous *FederateAmbassador::startUpdates* and can result from unpublication of the object class or lack of subscription interest among the rest of the federates. In the later case, the federate may continue updating the attributes in question, but is not required to do so.

Note that these callbacks do not occur synchronously with respect to the

(un)publication service, but rather are scheduled at a later time for delivery via the *RTIambassador::tick* service.

RETURN VALUES

A non-exceptional return from *RTIambassador::publishObjectClass* indicates that the given set of attributes has been published for the object class, possibly replacing an existing set of published attributes. The federate is then eligible to create objects of the given object class and to acquire instances of the specified attributes via ownership management services.

A non-exceptional return from *RTIambassador::unpublishObjectClass* indicates that the object class and all of its descendents have been unpublished, or that the object class is a derivative of *Manager* and the unpublication mechanism has silently refused to unpublish it. The federate may no longer create objects or acquire attributes of the given object class or any of its subclasses.

EXCEPTIONS

RTI::AttributeNotDefined - One or more attribute handles in the attribute-handle set is not valid in the context of the specified object class.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::FederateOwnsAttributes - The federate holds ownership tokens for object attributes that would be affected by the unpublication request. (not thrown in 1.0)

RTI::ObjectClassNotDefined - The object class handle is not valid in the context of the current federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::publishInteractionClass, *RTIambassador::registerObject*
RTIambassador::subscribeObjectClassAttribute,
RTIambassador::updateAttributeValues,
RTIambassador::requestAttributeOwnershipAcquisition,
FederateAmbassador::startUpdates, *FederateAmbassador::stopUpdates*
RTI::AttributeHandleSet

3.2.2 Publish Interaction Class

NAME

publishInteractionClass, *unpublishInteractionClass* -- convey the intention of the federate to begin (or cease) generating interactions of a given interaction class

HLA INTERFACE SPECIFICATION SERVICE

3.2 - *Declaration Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::publishInteractionClass (
  RTI::InteractionClassHandle theInteraction
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )

void
RTIambassador::unpublishInteractionClass (
  RTI::InteractionClassHandle theInteraction
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theInteraction

handle of the interaction class to be (un)published

DESCRIPTION

The *FederateAmbassador::publishInteractionClass* (*FederateAmbassador::unpublishInteractionClass*) method should be invoked by the federate when it wishes to notify the RTI that it will begin (end) the generation of the specified class of interactions. Attempts to generate an interaction whose class is unpublished will result in an *RTI::InteractionClassNotPublished* exception.

Publication of an interaction class does not imply the publication of subclasses of that class; each specific subclass must be explicitly published. On the other hand, unpublication of an interaction class also results in the unpublication of any of its published subclasses. The interaction class specified as the argument to

RTIambassador::unpublishInteractionClass need not actually be published; a federate could, for example, unpublish all interactions by unpublishing *ROOT_INTERACTION_CLASS_HANDLE*, the handle of the RTI-defined interaction class from which all federation-defined interaction classes are implicitly derived.

The federate may receive a *FederateAmbassador::startInteractionGeneration* callback for a published interaction class. Until such time, the federate may assume that no other federates have a subscription interest in the interaction class in question and that it is not necessary to generate that class of interaction. A *FederateAmbassador::stopInteractionGeneration* callback, which can result from an unpublication or cessation of subscription interest, negates the effect of *FederateAmbassador::startInteractionGeneration*.

Note that these callbacks do not occur synchronously with respect to the (un)publication service, but rather are scheduled at a later time for delivery via the *RTIambassador::tick* service.

RETURN VALUES

A non-exceptional return from *RTIambassador::publishInteractionClass* indicates that the specified interaction class has been added to the set of interactions published by the federate; the federate may begin generating interactions of this class.

A non-exceptional return from *RTIambassador::unpublishInteractionClass* indicates that the specified interaction class and all of its descendents have been removed from the set of interactions published by the federate; the federate may no longer generate interactions of these classes.

EXCEPTIONS

RTI::InteractionClassNotDefined - The specified interaction class handle is not valid within the context of the current federation execution.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::publishObjectClass, RTIambassador::sendInteraction, FederateAmbassador::subscribeInteractionClass, FederateAmbassador::startInteractionGeneration, FederateAmbassador::stopInteractionGeneration

3.2.3 Subscribe Object Class Attribute

NAME

subscribeObjectClassAttribute, unsubscribeObjectClassAttribute - declare or withdraw federate interest in receiving updates for a set of attributes

HLA INTERFACE SPECIFICATION SERVICE

3.3 - *Declaration Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::subscribeObjectClassAttribute (
  RTI::ObjectClassHandle theClass
  const RTI::AttributeHandleSet& attributeList
)
 throw (
  RTI::ObjectClassNotDefined,
  RTI::AttributeNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )

void
RTIambassador::unsubscribeObjectClassAttribute (
  RTI::ObjectClassHandle theClass
)
 throw (
  RTI::ObjectClassNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

Variants supporting subscription regions are not implemented in 1.0:

```
void
RTIambassador::subscribeObjectClassAttribute (
  RTI::ObjectClassHandle theClass
  RTI::AttributeHandle theAttribute
  RTI::Region theRegion
)
 throw (
  RTI::ObjectClassNotDefined,
  RTI::AttributeNotDefined,
  RTI::RegionNotKnown,
  RTI::FederateNotExecutionMember,
```

```
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError,
  RTI::UnimplementedService
 )

void
RTIambassador::unsubscribeObjectClassAttribute (
  RTI::ObjectClassHandle theClass
  RTI::Region theRegion
)
 throw (
  RTI::ObjectClassNotDefined,
  RTI::RegionNotKnown,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError,
  RTI::UnimplementedService
 )
```

ARGUMENTS

theClass

class handle of the object class whose subscription status will be affected by the call

attributeList

of attributes of the object class to subscribe to. The caller is responsible for freeing the storage associated with this list and may do so at any time after the completion of the call.

DESCRIPTION

These two methods are used by the federate to manipulate the types of data it wishes to have presented to the federate ambassador.

All examples in this section refer to the following class hierarchy (transport and ordering policies not shown):

(class *Beverage* (attribute *Volume*) (attribute *Price*) (class *CarbonatedBeverage* (attribute *Carbonation*) (class *Soda* (attribute *Caffeine*) (attribute *BrandName*))))

Subscription to an object class does not imply subscription to any subclasses of that object class; however, instances of non-subscribed subclasses will be promoted to the most specific subscribed object class for discovery by the federate ambassador. In doing so, the object manager filters out attributes that are not valid in the context of the discovered object class. For example, if a federate subscribes to all attributes of class *CarbonatedBeverage* and another federate creates an instance of *Soda*, this instance will be discovered as a *CarbonatedBeverage*, and the soda-specific attributes (*Caffeine* and *BrandName*) will be filtered out of the set of attributes presented to *FederateAmbassador::reflectAttributeValues*. Attribute subscriptions are not cumulative with respect to class hierarchies; for example, if a federate subscribes to object class *CarbonatedBeverage* with attributes *Carbonation* and *Volume* and object class *Beverage* with attributes *Volume* and *Price*, attribute updates for *CarbonatedBeverage* will not present the *Price* attribute to the federate ambassador, as it only uses the attribute subscription set for the most-specific

subscribed object class.

If *RTIambassador::subscribeObjectClassAttribute* is invoked with an object class that is already subscribed, the new attribute set replaces the existing subscribed attribute set. Subscription does not affect objects that have already been discovered by the federate ambassador; for example, if a federate initially subscribes to only *Beverage* but later subscribes to *Soda*, *Soda* instances that have previously been discovered as *Beverage* will not be rediscovered as the more specific object class.

Objects will not be discovered by the federate ambassador until an attribute update is received following the subscription of a relevant object class. Federates may wish to utilize the *RTIambassador::requestObjectAttributeValueUpdate* service to explicitly request an attribute update for pre-existing objects, especially if the attributes in question are only updated sporadically.

RTIambassador::unsubscribeObjectClassAttribute removes the specified object class and any of its subclasses from the set of object classes that will be presented to the federate ambassador. The specified object class need not actually be subscribed by the federate; for example, the federate may wish to unsubscribe all object classes by unsubscribing *ROOT_OBJECT_CLASS_HANDLE*, the handle of the RTI-defined object class from which all federation-defined object classes are implicitly derived.

Removal of an object class from the subscription set results in the removal of all instances of that class from the set of objects known to the federate ambassador. The federate ambassador will be informed of such through invocations of the "FederateAmbassador::removeObject" method (this notification is not delivered synchronously with respect to the unpublication method, but is queued up for later processing by the "RTIambassador::tick" service.) If the federate holds any attribute ownership tokens for objects removed from the federate ambassador, the object manager will automatically resolve ownership of these tokens; see the discussion of *RELEASE_ATTRIBUTES* in the *RTIambassador::resignFederationExecution* documentation for a detailed description of this process.

Object classes derived from the Management Object Model (MOM)-defined *Manager* class are not treated as a special case as they are by the publication services.

RETURN VALUES

A non-exceptional return from *RTIambassador::subscribeObjectClassAttribute* indicates that the federate has subscribed to the given object class and attribute set, replacing the existing attribute subscription set, if any. Future attribute updates of the subscribed attributes will be presented to the federate ambassador via the *FederateAmbassador::reflectAttributeValues* method.

A non-exceptional return from *RTIambassador::unsubscribeObjectClassAttribute* indicates that the given object class and any of its subclasses has been removed from the set of subscribed classes. All discovered instances of the affected classes have been queued for deletion from the set of objects known by the federate ambassador, and updates for the affected attributes will no longer be presented to the federate ambassador via the *FederateAmbassador::reflectAttributeValues* method. The object manager will attempt to divest attribute ownership tokens of any removed objects "behind the scenes".

EXCEPTIONS

RTI::ObjectClassNotDefined - The specified object class handle is not valid within the context of the current federation execution.

RTI::AttributeNotDefined - One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::publishObjectClass, FederateAmbassador::discoverObject, FederateAmbassador::removeObject, FederateAmbassador::reflectAttributeValues, RTIambassador::subscribeInteractionClass, RTIambassador::requestObjectAttributeValueUpdate, RTI::AttributeHandleSet

3.2.4 Subscribe Interaction Class

NAME

subscribeInteractionClass, unsubscribeInteractionClass - declare or withdraw federate interest in receiving a given class of interactions

HLA INTERFACE SPECIFICATION SERVICE

3.4 - *Declaration Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::subscribeInteractionClass (
  RTI::InteractionClassHandle theClass
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateLoggingServiceCalls,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )

void
RTIambassador::unsubscribeInteractionClass (
  RTI::InteractionClassHandle theClass
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

Variants supporting subscription regions are not implemented in 1.0:

```
void
RTIambassador::subscribeInteractionClass (
  RTI::InteractionClassHandle theClass
  RTI::Region theRegion
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::RegionNotKnown,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError,
  RTI::UnimplementedService
```

```
)

void
RTIambassador::unsubscribeInteractionClass (
 RTI::InteractionClassHandle theClass
 Region theRegion
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::RegionNotKnown,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError,
  RTI::UnimplementedService
 )
```

ARGUMENTS

theClass

interaction class handle of the interaction class whose subscription status will be affected by the call

DESCRIPTION

These two methods are used by the federate to manipulate the set of interaction classes whose instances will be presented to the federate via the *FederateAmbassador*.

Subscription to an interaction class does not imply subscription to any subclasses of that interaction class; however, instances of non-subscribed subclasses will be promoted to the most specific subscribed interaction class for presentation to the federate ambassador. For example, if a federate subscribes to interaction class *Foo* but not its subclass *Bar*, the interaction manager will promote incoming *Bar* interactions to *Foo* for presentation to *FederateAmbassador::receiveInteraction*; this includes the filtering out of parameters that are specific to class *Bar.*

Unlike object attributes, it is not possible to subscribe to a subset of interaction parameters; subscription to an interaction class implies subscription to every parameter of that interaction class.

Unsubscription of an interaction class also unsubscribes any subclasses of the interaction class. The specified interaction class need not actually be subscribed by the federate; for example, the federate may wish to unsubscribe all interaction classes by calling *RTIambassador::unsubscribeInteractionClass* with an argument of *ROOT_INTERACTION_CLASS_HANDLE*, the handle of the RTI-defined interaction class from which all federation-defined interaction classes are implicitly defined.

RETURN VALUES

A non-exceptional return from *RTIambassador::subscribeInteractionClass* indicates a successful subscription to the specified interaction class; future receipts of instances of the interaction class or any of its subclasses will be presented to the *FederateAmbassador::receiveInteraction* method (after any necessary class promotion and parameter filtering is done.)

A non-exceptional return from *RTIambassador::unsubscribeInteractionClass*

indicates a successful unsubscription of the specified interaction class and all of its subclasses; future receipts of instances of the affected interaction classes will not be presented to the federate ambassador.

EXCEPTIONS

RTI::InteractionClassNotDefined - The interaction class handle is not valid within the context of the current federation execution.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::publishInteractionClass, FederateAmbassador::receiveInteraction, RTIambassador::subscribeObjectClassAttribute, FederateAmbassador::startInteractionGeneration, FederateAmbassador::stopInteractionGeneration

3.2.5 Control Updates +

NAME

startUpdates, stopUpdates - informs the federate that it should begin (or cease) updating a given set of attributes of a given class

HLA INTERFACE SPECIFICATION SERVICE

3.5 - *Declaration Management* (RTI initiated)

SYNOPSIS

```
virtual
void
FederateAmbassador::startUpdates (
 RTI::ObjectClassHandle theClass
 const RTI::AttributeHandleSet& theAttributes
)
 throw (
  RTI::ObjectClassNotPublished,
  RTI::AttributeNotPublished,
  RTI::FederateInternalError
 )

virtual
void
FederateAmbassador::stopUpdates (
 RTI::ObjectClassHandle theClass
 const RTI::AttributeHandleSet& theAttributes
)
 throw (
  RTI::ObjectClassNotPublished,
  RTI::AttributeNotPublished,
  RTI::FederateInternalError
 )
```

ARGUMENTS

theClass

class whose attributes the federate should begin (or cease) updating.

theAttributes

set of attributes of the given class that the federate should begin (or cease) updating. The caller maintains ownership of the storage space associated with this set; the federate ambassador should make a copy if it wishes to retain this information after completion of the call.

DESCRIPTION

RTI invokes this callback to notify the federate that it should (or should not) generate updates of the specified set of attributes of the specified class. Such a notification applys only the specific object class given and is not intended to affect the update-generation status of any subclasses; if RTI wishes the federate to begin or cease update generation of a hierarchy of classes it will issue an explicit notification for each class in the hierarchy.

The federate will receive a *FederateAmbassador::startUpdates* notification when it

has published an attribute for which one or more other federates have declared a subscription interest. The *FederateAmbassador::stopUpdates* notification is issued when one of these conditions fails to hold, i.e. the federate has unpublished the attribute or all other federates have withdrawn their subscription interests.

The federate may still update any published attributes for which it has not received a *FederateAmbassador::startUpdates* notification; these callbacks should be seen merely as suggestions.

RETURN VALUES

A non-exceptional return from *FederateAmbassador::startUpdates* (*FederateAmbassador::stopUpdates*) indicates that the federate recognizes the object and has acknowledged the request to begin (cease) updating the specified set of attributes.

Exceptions thrown from these methods will cause any entry to be made in the federate's RTI log and otherwise ignored.

EXCEPTIONS

RTI::ObjectClassNotPublished - The object class handle is not recognized or the object class is not published by the federate.

RTI::AttributeNotPublished - One or more of the attribute handles is not recognized or the attribute is not published by the federate.

RTI::FederateInternalError - An error internal to the federate has occurred; this exception will result in an entry being made to the federate's RTI log.

SEE ALSO

FederateAmbassador::startInteractionGeneration, *RTIambassador::publishObjectClass*

3.2.6 Control Interactions +

NAME

startInteractionGeneration, stopInteractionGeneration - informs the federate that is should begin (or cease) the generation of a specified class of interaction

HLA INTERFACE SPECIFICATION SERVICE

3.6 - *Declaration Management* (RTI initiated)

SYNOPSIS

```
virtual
void
FederateAmbassador::startInteractionGeneration (
  RTI::InteractionClassHandle theHandle
)
 throw (
   RTI::InteractionClassNotPublished,
   RTI::FederateInternalError
 )

virtual
void
FederateAmbassador::stopInteractionGeneration (
  RTI::InteractionClassHandle theHandle
)
 throw (
   RTI::InteractionClassNotPublished,
   RTI::FederateInternalError
 )
```

ARGUMENTS

theHandle

class of interactions the federate should begin (or cease) the generation of.

DESCRIPTION

RTI invokes this callback to notify the federate that it should (or should not) generate interactions of the specified class. Such a notification applys only the specific interaction class given and is not intended to affect the update-generation status of any subclasses; if RTI wishes the federate to begin or cease update generation of a hierarchy of classes it will send an explicit notification for each class in the hierarchy.

The federate will receive a *FederateAmbassador::startInteractionGeneration* notification when it has published an interaction class for which one or more other federates have declared a subscription interest. The *FederateAmbassador::stopInteractionGeneration* notification is issued when one of these conditions fails to hold, i.e. the federate has unpublished the interaction class or all other federates have withdrawn their subscription interests.

The federate may still generate interactions of any published class for which it has not received a *FederateAmbassador::startInteractionGeneration* notification; these callbacks should be seen merely as suggestions.

RETURN VALUES

A non-exceptional return indicates that the federate recognizes the interaction class handle and will begin (or cease) the generation of the specified interaction class.

Exceptions thrown from this method will be entered into the federate's RTI log and ignored.

EXCEPTIONS

RTI::InteractionClassHandle - The interaction class handle is not recognized or the interaction class is not published by the federate.

RTI::FederateInternalError - An error internal to the federate has occurred; this exception will result in an entry being made to the federate's RTI log.

SEE ALSO

FederateAmbassador::startUpdates, RTIambassador::publishInteractionClass

3.3 Object Management

Table 3-25: Object Management Services

Section	Service Title	Service Implemented
4.1	Request ID	Yes
4.2	Register Object	Yes
4.3	Update Attribute Values	Yes
4.4	Discover Object †	Yes
4.5	Reflect Attribute Values †	Yes
4.6	Send Interaction	Yes
4.7	Receive Interaction †	Yes
4.8	Delete Object	Yes
4.9	Remove Object †	Yes
4.10	Change Attribute Transportation Type	Yes
4.11	Change Attribute Order Type	Yes
4.12	Change Interaction Transportation Type	Yes
4.13	Change Interaction Order Type	Yes
4.14	Request Attribute Value Update	Yes
4.15	Provide Attribute Value Update†	Yes
4.16	Retract	Yes
4.17	Reflect Retraction †	Yes

3.3.1 Request ID

NAME

requestID - obtain a range of unique IDs for use in registering objects with the federation

HLA INTERFACE SPECIFICATION SERVICE

4.1 - *Object Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::requestID (
  RTI::ObjectIDcount idCount
  RTI::ObjectID& firstID
  RTI::ObjectID& lastID
)
 throw (
  RTI::TooManyIDsRequested,
  RTI::IDsupplyExhausted,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

idCount
: number of IDs to reserve

firstID
: the first ID in the range of reserved IDs

lastID
: the last ID in the range of reserved IDs

DESCRIPTION

The federate invokes this method to obtain a set of unique object IDs that may be used in subsequent calls to *RTIambassador::registerObject*. These IDs are taken from the set of unique IDs that is assigned to every federate in a federation execution; thus object IDs are unique within the federate and within the federation execution at a given time (although the same object ID may be used by different federates at mutually exclusive times throughout a given federation execution.) IDs are not recycled upon the deletion of their associated object.

The number of unique IDs available to a federate is configurable via the *MAX_OBJECTS_PER_FEDERATE* entry in the *$RTI_CONFIG/RTI.rid* file.

RETURN VALUES

Upon a non-exceptional completion, *firstID* and *lastID* define the endpoints of an inclusive range of object IDs that may be used by the federate for the registration of

new federation objects.

WINDOWS NT NOTES

On Windows NT, the path of the RTI configuration file is *%RTI_CONFIG%.rid*.

EXCEPTIONS

RTI::TooManyIDsRequested - The request cannot be granted with a single continuous range of object IDs; try breaking the request up into multiple smaller requests. (Not thrown in RTI 1.0.)

RTI::IDsupplyExhausted - The federate has exhausted its supply of unique object IDs.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::registerObject

3.3.2 Register Object

NAME

registerObject - associate an object class with an object ID to create a new object in the federation execution

HLA INTERFACE SPECIFICATION SERVICE

4.2 - *Object Management* (federate initiated)

SYNOPSIS

```
void
RTIambassador::registerObject (
  RTI::ObjectClassHandle theClass
  RTI::ObjectID theObject
)
 throw (
  RTI::ObjectClassNotDefined,
  RTI::ObjectClassNotPublished,
  RTI::InvalidObjectID,
  RTI::ObjectAlreadyRegistered,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theClass

object class to associate with the specified object ID (this class must be published by the federate.)

theObject

object ID to associate with the specified class (should have been previously obtained via a call to *RTIambassador::requestID*.)

DESCRIPTION

This service associates the specified object class with the specified object ID, effectively creating a new object within the federation. Ownership tokens for attributes being published by the federate will be initially owned by the federate; any other ownership tokens will be initially unowned (specifically, they are tracked by the RTI internally in the process registering the object) and are available for acquisition by any federate. The initial transportation mechanism and update ordering policy for a given attribute are set to the values defined in the FOM file (*$RTI_CONFIG/[federation name].fed*.) *RTIambassador::registerObject* does not notify the federation of the existence of the newly created object; the object will not be discovered by other federates until an attribute update notification is sent out.

RETURN VALUES

A non-exceptional return indicates that the object has been successfully registered with RTI and the federate may begin generating updates for the attributes it publishes.

WINDOWS NT NOTES

On Windows NT, the path of the FOM file is *%RTI_CONFIG%federation].fed.*

=head1 EXCEPTIONS

RTI::ObjectClassNotDefined - The object class handle is not valid in the context of the current federation execution.

RTI::ObjectClassNotPublished - The specified object class is not published by the federate.

RTI::InvalidObjectID - The specified object ID has not been reserved for use by the federate.

RTI::ObjectAlreadyRegistered - An object has already been registered with the specified object ID.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::requestID, RTIambassador::publishObjectClass, RTIambassador::updateAttributeValues, RTIambassador::deleteObject, RTIambassador::changeAttributeTransportType, RTIambassador::changeAttributeOrderType, FederateAmbassador::discoverObject

3.3.3 Update Attribute Values

NAME

updateAttributeValues - notify the federation of a change in value of one or more attributes of an object

HLA INTERFACE SPECIFICATION SERVICE

4.3 - *Object Management* (federate initiated)

SYNOPSIS

```
RTI::EventRetractionHandle
RTIambassador::updateAttributeValues (
 RTI::ObjectID theObject
 const RTI::AttributeHandleValuePairSet& theAttributes
 RTI::FederationTime theTime
 const RTI::UserSuppliedTag theTag
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotDefined,
  RTI::AttributeNotOwned,
  RTI::InvalidFederationTime,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theObject

object ID of the instance whose attributes are being updated.

theAttributes

set of handles of the attributes being updated and their associated values. The caller is responsible for freeing the storage space associated with this set and may do so at its leisure.

theTime

federation time at which the new attribute values should take effect.

theTag

string value that is passed to the invocations *FederateAmbassador::reflectAttributeValues* on the remote federates; this may contain a textual description of what caused the change in state or any other data that is meaningful for a particular federation. The caller is responsible for freeing the storage associated with this string and may do so at its leisure.

DESCRIPTION

This method notifies the other federates in the federation execution of a change in the specified attribute values of the specified object. This may result in one or more invocations of *FederateAmbassador::reflectAttributeValues* on one or more remote

federates (after any class promotion and attribute filtering necessary to meet a given federate's subscription criteria is done.)

The transportation mechanism and ordering policy used for a given attribute are defined by the federate's FED file (*$RTI_CONFIG/[federation name].fed*) or may be specified dynamically on a per-instance basis using *RTIambassador::changeAttributeTransportType* and *RTIambassador::changeAttributeOrderType* (keep in mind that the attribute-specific ordering policy is only considered if the federate is time regulating, otherwise all attribute updates are sent receive-ordered.) Attributes with different transportation mechanisms and/or ordering policies may be grouped together in a single *RTIambassador::updateAttributeValues* invocation; the object manager will send out various subsets of the specified attributes in physically separate updates as necessary.

The federate must hold the ownership tokens for any attributes it attempts to update (and therefore must also be publishing the attributes.)

RETURN VALUE

The *RTI::EventRetractionHandle* returned may be used as the argument to *RTIambassador::retract* to withdraw the update notification.

EXCEPTIONS

RTI::ObjectNotKnown - The specified object ID is not valid within the current federation execution or is not known to the federate.

RTI::AttributeNotDefined - One or more of the specified attributes is not valid in the context of the specified object (valid attributes are still sent out before this exception is raised.)

RTI::AttributeNotOwned - The federate does not hold the ownership token for one or more of the specified attributes (valid attributes are still sent out before this exception is raised.)

RTI::InvalidFederationTime - The time value specified is not a legal time for a time-stamp-ordered update to be posted by the federate, i.e. it is less than the federate's logical time plus its lookahead. (Not thrown in 1.0.)

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::reflectAttributeValues, RTIambassador::publishObjectClass, RTIambassador::queryAttributeOwnership, RTIambassador::changeAttributeTransportType, RTIambassador::changeAttributeOrderType, RTIambassador::registerObject, FederateAmbassador::discoverObject, RTIambassador::turnRegulationOn, RTI::AttributeHandleValuePairSet, RTIambassador::tick RTIambassador::retract

3.3.4 Discover Object +

NAME

discoverObject - inform the federate of the existence of an object in the federation

HLA INTERFACE SPECIFICATION SERVICE

4.4 - *Object Management* (RTI initiated)

SYNOPSIS

```
virtual
void
FederateAmbassador::discoverObject (
 RTI::ObjectID theObject
 RTI::ObjectClassHandle theObjectClass
 RTI::FederationTime theTime
 const RTI::UserSuppliedTag theTag
 RTI::EventRetractionHandle theHandle
)
 throw (
  RTI::CouldNotDiscover,
  RTI::ObjectClassNotKnown,
  RTI::InvalidFederationTime,
  RTI::FederateInternalError
 )
```

ARGUMENTS

theObject

object ID of the object being discovered.

theObjectClass

discovered class of the object (this may differ from its actual class -- see below.)

theTime

time at which the object is discovered (not necessarily the time at which the object was created.)

theTag

string value that was passed to the *RTIambassador::updateAttributeValues* that triggered the object discovery. This may contain a textual description of the reason for the attribute update, or any other data that is meaningful for a particular federation. The caller maintains ownership of the storage associated with this string; the federate ambassador should make a copy if it wishes to retain this information after the completion of the call.

theHandle

event handle that may later be used to retract the discovery of this object.

DESCRIPTION

This method is invoked to inform the federate of a newly discovered object that meets the federate's subscription criteria. This occurs upon the receipt of an attribute update for an object that has not previously been discovered by the federate; there is no notification of new objects that have been registered using

RTIambassador::registerObject but have not been updated. Discoveries always immediately precede a *FederateAmbassador::reflectAttributeValues* for the discovered object (no intervening *RTIambassador::tick* is required); an object is discovered exactly when one or more of its attribute updates becomes eligible for presentation to the federate (see *RTIambassador::reflectAttributeValues* and *RTIambassador::tick* for a more thorough discussion of when this occurs.)

If the actual class of an object is a subclass of an object class subscribed by the federate, the object is promoted to the subscribed class for discovery by the federate. For example, if a federate subscribes to class *Foo* but not its subclass *Bar,* instances of class *Bar* will be discovered as *Foo* by the federate.

Only objects that have been discovered by a federate are eligible for updates via invocations of *FederateAmbassador::reflectAttributeValues*.

RETURN VALUES

A non-exceptional return indicates that the federate understands the discovery notification.

An exceptional return will cause an error message to be written to the federate's RTI log file; the object will still be considered discovered and subject to future update notifications.

EXCEPTIONS

RTI::CouldNotDiscover - The federate could not discover the specified object (it's not particularly clear when this should be thrown.)

RTI::ObjectClassNotKnown - The objet class handle is not valid in the context of the current federation execution or is not subscribed by the federate.

RTI::InvalidFederationTime - The federation time is not valid, i.e. a time-stamped order update has been delivered to a time-constrained federation in the federate's past. (In 1.0 there's no way for the federate to tell whether the discovery is the result of a time-stamp ordered update, so it's unclear when this exception should be raised.)

RTI::FederateInternalError - An error internal to the federate has occurred.

SEE ALSO

RTIambassador::registerObject RTIambassador::subscribeObjectClass, RTIambassador::updateAttributeValues, FederateAmbassador::reflectAttributeValues, FederateAmbassador::reflectRetration, FederateAmbassador::removeObject

3.3.5 Reflect Attribute Values +

NAME

reflectAttributeValues - inform the federate of a change in one or more attribute values of a previously-discovered object in the federation

HLA INTERFACE SPECIFICATION SERVICE

4.5 - *Object Management* (RTI initiated)

SYNOPSIS

```
virtual
void
FederateAmbassador::reflectAttributeValues (
 RTI::ObjectID theObject
 const RTI::AttributeHandleValuePairSet& theAttributes
 RTI::FederationTime theTime
 const RTI::UserSuppliedTag theTag
 RTI::EventRetractionHandle theHandle
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotKnown,
  RTI::InvalidFederationTime,
  RTI::FederateInternalError
 )
```

ARGUMENTS

theObject

object whose attributes are being updated.

theAttributes

set of attributes being updated and their corresponding new values. The caller maintains ownership of the storage associated with this set; the federate ambassador should make a copy if it wishes to retain this information after the completion of the call.

theTime

time at which the updated values are effective.

theTag

string value that was passed to the invocation of *RTIambassador::updateAttributeValues* that triggered this reflection. This may contain a textual description of the reason for the attribute update, or any other data that is meaningful for a particular federation. The caller maintains ownership of the storage associated with this string; the federate ambassador should make a copy if it wishes to retain this information after the completion of the call.

theHandle

event handle that may later be used to retract the specified update.

DESCRIPTION

This callback is invoked to inform the federate that the given set of attributes have

changed values, effective at the specified time. This occurs as a result of a call to *RTIambassador::updateAttributeValues* made by a remote federate (a single update may cause more than one reflection per federate, possibly at different physical and logical times, if the attributes updated differ in transport mechanisms or ordering policies.)

If the federate is time-constrained, attribute updates designated by the sender as time-stamp-ordered are not presented to the federate until it has requested to advance its logical time to a time equal to or beyond the time-stamp of the update and it can be guaranteed that no time-stamp-ordered updates or interactions with a lower time-stamp will be received, i.e. the federation's lower-bound time stamp is greater than or equal to the update's time-stamp. Time-stamp-ordered updates and interactions are presented to the federate in strictly non-decreasing time order (although delivery of messages with the same time-stamp occurs in non-deterministic order.)

If the federate is not time-constrained, updates and interactions are presented to the federate in the order in which they are received, regardless of their time-stamp and ordering policies.

RETURN VALUES

A non-exceptional return indicates that the federate understands the attribute update notification.

An exceptional return will cause an error message to be written to the federate's RTI log file; the attribute values are still considered to have been reflected.

EXCEPTIONS

RTI::ObjectNotKnown - The object ID is has not previously been discovered by the federate.

RTI::AttributeNotKnown - One or more attribute handles are not valid in the context of the current federation execution or the attributes are not subscribed by the federate.

RTI::InvalidFederationTime - The federation time is not valid, i.e. a time-stamped-ordered update has been delivered to a time-constrained federation in the federate's past. (In 1.0 there's no way for the federate to tell whether the reflection is the result of a time-stamp-ordered update, so it's unclear when this exception should be raised.)

RTI::FederateInternalError - An error internal to the federate has occurred.

SEE ALSO

RTIambassador::updateAttributeValues, RTIambassador::setTimeConstrained, RTIambassador::subscribeObjectClass, FederateAmbassador::discoverObject, RTI::AttributeHandleValuePairSet, FederateAmbassador::reflectRetration, RTIambassador::timeAdvanceRequest, FederateAmbassador::receiveInteraction

3.3.6 Send Interaction

NAME

sendInteraction - notify the federation of an action taken by an object, possibly directed towards another object

HLA INTERFACE SPECIFICATION SERVICE

4.6 - *Object Management* (federate initiated)

SYNOPSIS

```
RTI::EventRetractionHandle
RTIambassador::sendInteraction (
 RTI::InteractionClassHandle theInteraction
 const RTI::ParameterHandleValuePairSet& theParameters
 RTI::FederationTime theTime
 const RTI::UserSuppliedTag theTag
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::InteractionClassNotPublished,
  RTI::InteractionParameterNotDefined,
  RTI::InvalidFederationTime,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theInteraction

class of interaction to send to the federation (must be a class published by the federate.)

theParameters

set handles of the parameters being updated and their associated values. The caller is responsible for freeing the storage space associated with this set and may do so at its leisure.

theTime

federation time at which the interaction occurs.

theTag

string value that is passed to the invocations *FederateAmbassador::receiveInteraction* on the remote federates; this make contain a textual description of the interaction, or any other data that is meaningful for a particular federation. The caller is responsible for freeing the storage associated with this string and may do so at its leisure.

DESCRIPTION

This service allows the federate to notify the federation of an action taken by some object in the federation or any other change in federate state that cannot be communicated via attribute value updates. This may result in a single invocation of

FederateAmbassador::receiveInteraction on one or more remote federates (after any class promotion and parameter filtering necessary to meet a given federate's subscription criteria is done.)

The transportation mechanism and ordering policy used for the interaction are defined by the federate's FED file (*$RTI_CONFIG/[federation name].fed*) or may be specified dynamically using *RTIambassador::changeInteractionTransportType* and *RTIambassador::changeInteractionOrderType* (keep in mind that the interaction class-specific ordering policy is only considered if the federate is time regulating, otherwise all interactions are sent receive-ordered.)

RETURN VALUES

The *RTI::EventRetractionHandle* returned may be used as the argument to *RTIambassador::retract* to withdraw the interaction.

EXCEPTIONS

RTI::InteractionClassNotDefined - The specified interaction class handle is not valid in the context of the current federation execution.

RTI::InteractionClassNotPublished - The specified interaction class is not currently published by the federate.

RTI::InteractionParameterNotDefined - One or more of the specified interaction parameter handles are not valid in the context of the specified interaction class.

RTI::InvalidFederationTime - The time value specified is not a legal time for a time-stamp-ordered update to be posted by the federate, i.e. it is less than the federate's logical time plus its lookahead. (Not thrown in 1.0.)

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::receiveInteraction, RTIambassador::publishInteractionClass, RTIambassador::changeInteractionTransportType, RTIambassador::changeInteractionOrderType, RTIambassador::turnRegulationOn, RTI::ParameterHandleValuePairSet, RTIambassador::tick RTIambassador::retract

3.3.7 Receive Interaction +

NAME

receiveInteraction - inform the federate of an interaction generated by another federate in the federation execution

HLA INTERFACE SPECIFICATION SERVICE

4.7 - *Object Management* (RTI initiated)

SYNOPSIS

```
virtual
void
FederateAmbassador::receiveInteraction (
 RTI::InteractionClassHandle      theInteraction
 const RTI::ParameterHandleValuePairSet& theParameters
 RTI::FederationTime theTime
 const RTI::UserSuppliedTag theTag
 RTI::EventRetractionHandle theHandle
)
 throw (
  RTI::InteractionClassNotKnown,
  RTI::InteractionParameterNotKnown,
  RTI::InvalidFederationTime,
  RTI::FederateInternalError
 )
```

ARGUMENTS

theInteraction

class of the interaction (must be a class subscribed by the federate or a subclass of a subscribed class.)

theParameters

set of parameters for this interaction and their associated values. The caller maintains ownership of the storage associated with this set; the federate ambassador should make a copy if it wishes to retain this information after the completion of the call.

theTime

time at which the interaction occurs.

theTag

string value that was passed to the invocation of *RTIambassador::sendInteraction* that triggered this reflection. This may contain a textual description of the interaction, or any other data that is meaningful for a particular federation. The caller maintains ownership of the storage associated with this string; the federate ambassador should make a copy if it wishes to retain this information after the completion of the call.

theHandle

event handle that may later be used to retract the specified interaction.

DESCRIPTION

This callback is invoked to inform the federate of an interaction generated by

another federate in the federation execution via *RTIambassador::sendInteraction*. Federates are only notified of interactions that meet their subscription criteria, as defined by *RTIambassador::(un)subscribeInteractionClass*.

If the federate is time-constrained, interactions designated by the sender as time-stamp-ordered are not presented to the federate until it has requested to advance its logical time past the time-stamp of the interaction and it can be guaranteed that no time-stamp-ordered updates or interactions with a lower time-stamp will be received, i.e. the federation's lower-bound time stamp is greater than or equal to the update's time-stamp. Time-stamp-ordered updates and interactions are presented to the federate in strictly non-decreasing time order (although delivery of messages with the same time-stamp occurs in non-deterministic order.)

If the federate is not time-constrained, updates and interactions are presented to the federate in the order in which they are received, regardless of their time-stamp and ordering policies.

RETURN VALUES

A non-exceptional return indicates that the federate understands the interaction notification.

An exceptional return will cause an entry to be made in the federate's RTI log file; the interaction is still considered to have been delivered to the federate.

EXCEPTIONS

RTI::InteractionClassNotKnown - The specified interaction class handle is not valid in the context of the current federation execution or is not subscribed by the federate.

RTI::InteractionParameterNotKnown - One or more of the specified parameters handles is not valid in the context of the specified interaction class.

RTI::InvalidFederationTime - The federation time is not valid, i.e. a time-stamped-ordered interaction has been delivered to a time-constrained federation in the federate's past. (In 1.0 there's no way for the federate to tell whether the receipt is the result of a time-stamp-ordered interaction, so it's unclear when this exception should be raised.)

RTI::FederateInternalError - An error internal to the federate has occurred.

SEE ALSO

RTIambassador::sendInteraction, RTIambasador::setTimeConstrained, RTIambassador::subscribeInteractionClass, RTIambassador::ParameterHandleValuePairSet, FederateAmbassador::reflectRetraction, RTIambassador::timeAdvanceRequest, FederateAmbassador::reflectAttributeValues

3.3.8 Delete Object

NAME

deleteObject - remove an object from the federation execution

HLA INTERFACE SPECIFICATION SERVICE

4.8 - *Object Management* (federate initiated)

SYNOPSIS

```
RTI::EventRetractionHandle
RTIambassador::deleteObject (
 RTI::ObjectID objectID
 RTI::FederationTime theTime
 const RTI::UserSuppliedTag theTag
)
 throw (
  RTI::ObjectNotKnown,
  RTI::DeletePrivilegeNotHeld,
  RTI::InvalidFederationTime,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

objectID

object to be deleted from the federation execution.

theTime

time at which the object deletion is to become effective.

theTag

string value that will be passed to resulting invocations of *FederateAmbassador::removeObject*. This may contain a description of the reason for the object deletion, or any other information that is meaningful for a particular federation. The caller is responsible for freeing the storage space associated with this string and may do so at its leisure.

DESCRIPTION

The federate invokes this service when it wishes to remove an object from the federation execution. To delete an object, a federate must hold the ownership token of the object's special "privilege to delete" attribute (referenced by *PRIVILEGE_TO_DELETE_HANDLE*.) This token is initially held by the federate that registered the object with the federation execution.

If the federate is time regulating and any of the instance's attributes are being sent time-stamp-ordered by the federate, the object deletion message will be designated for time-stamp-ordered delivery, otherwise it will be sent receive-ordered.

A successful invocation of this service will trigger *FederateAmbassador::removeObject* callbacks on federates that have discovered the

specified object.

RETURN VALUES

A non-exceptional return indicates that an object-deletion message has been sent to the other federates in the federation execution.

The *RTI::EventRetractionHandle* returned by this function can be used to reinstate the object via the *RTIambassador::retract* service.

EXCEPTIONS

RTI::ObjectNotKnown - The object ID is not valid in the context of the current federation execution or the object is not known to the federate.

RTI::DeletePrivilegeNotHeld - The federate does not hold the ownership token of the special "privilege to delete" attribute.

RTI::InvalidFederationTime - The The time value specified is not a legal time for a time-stamp-ordered update to be posted by the federate, i.e. it is less than the federate's logical time plus its lookahead. (Not thrown in 1.0.)

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::removeObject, RTIambassador::registerObject, RTIambassador::retract RTIambassador::queryAttributeOwnership

3.3.9 Remove Object +

NAME

removeObject - informs the federate that a specified object has been removed from the federation execution or no longer meets the interest criteria of the federate

HLA INTERFACE SPECIFICATION SERVICE

4.9 - *Object Management* (RTI initiated)

SYNOPSIS

enum RTI::ObjectRemovalReason *OUT_OF_REGION* = 1, *OBJECT_DELETED*, *NO_LONGER_SUBSCRIBED* ;

```
virtual
void
FederateAmbassador::removeObject (
 RTI::ObjectID theObject
 RTI::ObjectRemovalReason theReason
 RTI::FederationTime theTime
 const RTI:UserSuppliedTag theTag
 RTI::EventRetractionHandle theHandle
)
 throw (
  RTI::ObjectNotKnown,
  RTI::InvalidFederationTime,
  RTI::FederateInternalError
 )
```

A variation is invoked when the object is being removed because its class has been unsubscribed or it is no longer a constituent of any federate-subscribed region:

```
virtual
void
FederateAmbassador::removeObject (
 RTI::ObjectID theObject
 RTI::ObjectRemovalReason theReason
)
 throw (
  RTI::ObjectNotKnown,
  RTI::InvalidFederationTime,
  RTI::FederateInternalError
 )
```

ARGUMENTS

theObject

object ID of the object being removed (must be an object that has previously been discovered by the federate.)

theReason

value indicating the reason why the object is being removed from the federate.

theTime

time at which the removal is effective.

theTag
string value that was passed to the invocation of *RTIambassador::deleteObject* that triggered this notification. This may contain a textual description of the reason for deletion, or any other data that is meaningful for a particular federation. The caller maintains ownership of the storage associated with this string; the federate ambassador should make a copy if it wishes to retain this information after the completion of the call.

theHandle
event handle that may later be used to retract the specified removal.

DESCRIPTION

This callback notifies the federate that the specified object has been removed from the set of objects "known" by the federate; this can occur when an object is deleted from the federation execution, the object's class is unsubscribed by the federate, or the object no longer belongs to any subscribed region of the federate.

The federate will receive no further attribute reflections for objects that have been deleted (unless they are reinstated with a subsequent *FederateAmbassador::discoverObject*.) Any ownership tokens for attributes of this object held by the federate have been destroyed, transferred to another federate, or become unowned.

If the federate is time-constrained and the object removal is due to a time-stamp-ordered message from another federate (i.e. a deletion or an update that causes the the object to no longer be included in any federate-subscribed regions), the removal notification will be scheduled for presentation to the federate in time-stamped-order (see *RTIambassador::reflectAttributeValues* for a discussion of what this means), otherwise it will be eligible for immediate presentation (i.e. receive-ordered.)

RETURN VALUES

A non-exceptional return indicates that the federate understands the removal notification.

An exceptional return will cause an entry to be made in the federate's RTI log; the object will still be considered deleted from the federate.

EXCEPTIONS

RTI::ObjectNotKnown - The object ID does not correspond to an object previously discovered by the federate.

RTI::InvalidFederationTime - The federation time is not valid, i.e. a time-stamped-ordered deletion notification has been delivered to a time-constrained federation in the federate's past. (In 1.0 there's no way for the federate to tell whether the reflection is the result of a time-stamp-ordered update, so it's unclear when this exception should be raised.)

RTI::FederateInternalError - An error internal to the federate has occurred.

SEE ALSO

RTIambassador::deleteObject, FederateAmbassador::discoverObject, RTIambassador::unsubscribeObjectClassAttribute,

FederateAmbassador::reflectRetraction

3.3.10 Change Attribute Transportation Type

NAME

changeAttributeTransportType - change the transportation mechanism used by the federate for updates of a specified set of attributes of a specified object

HLA INTERFACE SPECIFICATION SERVICE

4.10 - *Object Management* (federate initiated)

SYNOPSIS

enum RTI::TransportType *RELIABLE* = 1, *BEST_EFFORT* ;

```
void
RTIambassador::changeAttributeTransportType (
 RTI::ObjectID theObject
 const RTI::AttributeHandleSet& theAttributes
 RTI::TransportType theType
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotDefined,
  RTI::AttributeNotOwned,
  RTI::InvalidTransportType,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theObject

object whose attributes the federate wishes to change the transportation mechanism of.

theAttributes

set of attributes that the federate wishes to change the transportation mechanism of. The caller is responsible for freeing the storage associated with this parameter and may do so at its leisure.

theType

enumerated value specifying the new transportation mechanism to be used for the updates of the specified attributes.

DESCRIPTION

The federate can utilize this service to dynamically specify the mechanism by which updates of the given set of object-attributes are delivered (the default value is determined by the federate initialization file, *$RTI_CONFIG/[federation name].fed*.) This change only affects the local federate and only affects the attributes of the specified class instance (not the object class itself.) The federate need not own a given attribute in order to change its transport mechanism.

The following transport options are available:

RELIABLE
: are guaranteed to be delivered to each of the federates currently joined in the federation. In 1.0 this is implemented using TCP sockets, with the federation executive serving as an "exploder" for federation messages. This means that reliable updates are high-latency, may cause the federate to block while sending, and have limited bandwidth available. However, it is necessary to use this transport for essential updates (e.g. missile detonations, collision notifications.)

BEST_EFFORT
: might not be delivered to one or more federates currently involved in federation execution. In 1.0 this is implemented using multicast datagrams. Best effort updates are low-latency and low-overhead and are therefore the preferred transport mechanism for non-essential types of updates (e.g. routine position notifications.)

RETURN VALUES

A non-exceptional exit indicates that future updates for the specified object-attributes will use the specified transport mechanism.

EXCEPTIONS

RTI::ObjectNotKnown - The object handle is not valid within the context of the federation execution or the object is not known by the federate.

RTI::AttributeNotDefined - One or more of the attribute handles is not valid within the context of the specified object.

RTI::AttributeNotOwned - Not thrown; the federate need not own a given attribute to change its transport type.

RTI::InvalidTransportType - The specified transport type was not a recognized enumerated value.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::changeAttributeOrderType,
RTIambassador::updateAttributeValues,
RTIambassador::changeInteractionTransportType, RTI::AttributeHandleSet

3.3.11 Change Attribute Order Type

NAME

changeAttributeOrderType - change the ordering policy used by the federate for updates of a specified set of attributes of a specified object

HLA INTERFACE SPECIFICATION SERVICE

4.11 - *Object Management* (federate initiated)

SYNOPSIS

enum RTI::OrderType *RECEIVE* = 1, *TIMESTAMP* ;

```
void
RTIambassador::changeAttributeOrderType (
  RTI::ObjectID theObject
  const RTI::AttributeHandleSet& theAttributes
  RTI::OrderType theType
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotDefined,
  RTI::AttributeNotOwned,
  RTI::InvalidOrderType,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theObject

object whose attributes the federate wishes to change the ordering policy of.

theAttributes

set of attributes that the federate wishes to change the ordering policy of. The caller is responsible for freeing the storage associated with this parameter and may do so at its leisure.

theType

enumerated value specifying the new ordering policy to be used for updates of the specified attributes.

DESCRIPTION

The federate can utilize this service to dynamically specify the ordering policy by which updates of the given set of object-attributes by the local federate are delivered (the default value is determined by the federate initialization file, *$RTI_CONFIG/[federation name].fed*.) This change only affects the local federate and only affects the attributes of the specified class instance (not the object class itself.) The federate need not own a given attribute in order to change its ordering policy.

Note that if the federate is not time regulating, all updates will be sent as receive-

order, regardless of the attributes' ordering policies.

The following ordering policies are available:

RECEIVE
: receipt of the update by a remote federate's transportation manager, it is immediately placed into its event queue for processing by *RTIambassador::tick*, regardless of the update's time-stamp. This policy minimizes overhead, but out-of-order delivery is unacceptable in some situations (of course, the federate may choose to do its own ordering based, e.g., on time information encoded in the user-specified tag portion of the update.)

TIMESTAMP
: a given remote federate is time constrained, the update is placed in its time-stamp-ordered message queue for delivery at the appropriate time, as determined by the update time-stamp. Federates that are not time constrained will treat the message as if it were receive-order. This policy incurs more overhead but will be necessary for certain types of simulations.

RETURN VALUES

A non-exceptional return indicates that future updates of the specified object-attributes will use the specified ordering policy.

EXCEPTIONS

RTI::ObjectNotKnown - The object handle is not valid within the context of the federation execution or the object is not known by the federate.

RTI::AttributeNotDefined - One or more of the attribute handles is not valid within the context of the specified object.

RTI::AttributeNotOwned - Not thrown; the federate need not own a given attribute to change its ordering policy.

RTI::InvalidOrderType - The specified ordering policy was not a recognized enumerated value.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::changeAttributeTransportType, *RTIambassador::updateAttributeValues*, *RTIambassador::tick*, *RTIambassador::turnRegulationOn*, *RTIambassador::setTimeConstrained*, *RTI::AttributeHandleSet*

3.3.12 Change Interaction Transportation Type

NAME

changeInteractionTransportType - change the transportation mechanism used by the federate for interactions of a specified class

HLA INTERFACE SPECIFICATION SERVICE

4.12 - *Object Management* (federate initiated)

SYNOPSIS

enum RTI::TransportType *RELIABLE* = 1, *BEST_EFFORT* ;

```
void
RTIambassador::changeInteractionTransportType (
 RTI::InteractionClassHandle theClass
 RTI::TransportType theType
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::InteractionClassNotPublished,
  RTI::InvalidTransportType,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theClass

class of interactions that the federate wishes to change the transportation mechanism of.

theType

enumerated value specifying the new transportation mechanism to be used for interactions of the specified class.

DESCRIPTION

The federate can utilize this service to dynamically specify the mechanism by which instances of a given interactions class are delivered (the default value is determined by the federate initialization file, *$RTI_CONFIG/[federation name].fed*.) This change only affects the local federate and only affects the specified interaction class (not subclasses of the interaction class.) The federate must be currently publishing an interaction class in order to change its transport mechanism.

The following transport options are available:

RELIABLE

are guaranteed to be delivered to each of the federates currently joined in the federation. In 1.0 this is implemented using TCP sockets, with the federation executive serving as an "exploder" for federation messages. This means that reliable interactions are high-latency, may cause the federate to block while sending, and have limited bandwidth available. However, it is

necessary to use this transport for essential interactions (e.g. missile detonations, collision notifications.)

BEST_EFFORT
might not be delivered to one or more federates currently involved in federation execution. In 1.0 this is implemented using multicast datagrams. Best effort interactions are low-latency and low-overhead and are therefore the preferred transport mechanism for non-essential types of interactions.

RETURN VALUES

A non-exceptional return indicates that future interactions of the specified class will be sent using the specified transportation mechanism.

EXCEPTIONS

RTI::InteractionClassNotDefined - The interaction class handle is not valid within the context of the current federation execution.

RTI::InteractionClassNotPublished - The federate is not currently publishing the given interaction class.

RTI::InvalidTransportType - The specified transport type was not a recognized enumerated value.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::changeInteractionOrderType, RTIambassador::sendInteraction, RTIambassador::changeAttributeTransportType, RTIambassador::publishInteractionClass

3.3.13 Change Interaction Order Type

NAME

changeInteractionOrderType - change the ordering policy used by the federate for interactions of a specified class

HLA INTERFACE SPECIFICATION SERVICE

4.13 - *Object Management* (federate initiated)

SYNOPSIS

enum RTI::OrderType *RECEIVE* = 1, *TIMESTAMP* ;

```
void
RTIambassador::changeInteractionOrderType (
  RTI::InteractionClassHandle theClass
  RTI::OrderType theType
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::InteractionClassNotPublished,
  RTI::InvalidOrderType,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theClass

class of interactions that the federate wishes to change the ordering policy of.

theType

enumerate value specifying the new ordering policy to be used for instances of the specified interaction class.

DESCRIPTION

The federate can utilize this service to dynamically specify the ordering policy by which instances of the given interaction class are delivered (the default value is determined by the federate initialization file, *$RTI_CONFIG/[federation name].fed*.) This change only affects the local federate and only affects the specified interaction class (not subclasses of the interaction class.) The federate must be currently publishing an interaction class in order to change its ordering policy.

Note that if the federate is not time regulating, all interactions will be sent as receive-order, regardless of the interactions' stated ordering policies.

The following ordering policies are available:

RECEIVE

receipt of the interaction by a remote federate's transportation manager, it is

immediately placed into its event queue for processing by *RTIambassador::tick*, regardless of the interaction's time-stamp.

TIMESTAMP
a given remote federate is time constrained, the interaction is placed in its time-stamp-ordered message queue for delivery at the appropriate time, as determined by the update time-stamp. Federates that are not time constrained will treat the interaction as if it were receive-order.

RETURN VALUES

A non-exceptional return indicates that future interactions of the specified class will be sent using the specified ordering policy.

EXCEPTIONS

RTI::InteractionClassNotDefined - The interaction class handle is not valid within the context of the current federation execution.

RTI::InteractionClassNotPublished - The federate is not currently publishing the given interaction class.

RTI::InvalidOrderType - The specified ordering policy was not a recognized enumerated value.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::changeInteractionTransportType, *RTIambassador::updateInteractionValues*, *RTI::InteractionHandleSet*

3.3.14 Request Attribute Value Update

NAME

requestObjectAttributeValueUpdate, *requestClassAttributeValueUpdate* - stimulate the generation of attribute updates for a given object or a given class of objects

HLA INTERFACE SPECIFICATION SERVICE

4.14 - *Object Management* (federate initiated)

SYNOPSIS

```
void
RTIambassador::requestObjectAttributeValueUpdate (
  RTI::ObjectID theObject
  const RTI::AttributeHandleSet& theAttributes
)
 throw (
  RTI::AttributeNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

A second method can be used to stimulate updates for all instances of a given object class:

```
void
RTIambassador::requestClassAttributeValueUpdate (
  RTI::ObjectClassHandle theClass
  const RTI::AttributeHandleSet& theAttributes
)
 throw (
  RTI::ObjectClassNotDefined,
  RTI::AttributeNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theObject
: object whose attributes are being requested.

theClass
: object class whose attributes are being requested.

theAttributes
: set of attributes of the given object or class for which updates are requested. The caller is responsible for freeing the storage space associated with this set and may do so at its leisure.

DESCRIPTION

This method solicits an update of the specified object-attributes from the federation. This will result in invocations of *FederateAmbassador::provideAttributeValueUpdate* on the federate(s) holding the ownership tokens of the specified object-attributes. Upon receipt of an attribute value update request, the remote federate should issue an update of the requested attributes.

If the request is for a class of objects, then updates of subclasses of the class are requested as well.

As a given object is not discovered by a federate until the receipt of an attribute update for that object, this service proves to be useful way for late-arriving federates to discover objects already existing in the federation (particularly if updates for these objects are ordinarily made infrequently.)

One or more updates can result from a single attribute update request (even for a single object instance), as different subsets of the requested attributes may be owned by different federates. There is no guarantee that updates for all the requested attributes will be received. If an attribute's ownership token no longer exists or is not held by any federate (or if the federate holding the token ignores the attribute update request!), the requesting federate will simply not receive an attribute update for the attribute; no negative acknowledgement is provided.

RETURN VALUES

A non-exceptional return indicates that the specified object-attribute updates have been solicited from the federation; the federate will be notified of any results via its *FederateAmbassador::reflectAttributeValues* callback (this does not occur synchronously with respect to the request method, but will occur at a later time in response to an invocation of the *RTIambassador::tick*.)

EXCEPTIONS

RTI::AttributeNotDefined - One or more of the attribute handles is not valid within the context of the specified object or object class.

RTI::ObjectClassNotDefined - The specified object class handle is not valid within the context of the current federation execution.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::provideAttributeValueUpdate, *RTIambassador::discoverObject*, *FederateAmbassador::reflectAttributeValues*

3.3.15 Provide Attribute Value Update +

NAME

provideAttributeValueUpdate - callback invoked by RTI to solicit an attribute-value update from the federate

HLA INTERFACE SPECIFICATION SERVICE

4.15 - *Object Management* (RTI initiated)

SYNOPSIS

```
virtual
void
FederateAmbassador::provideAttributeValueUpdate (
 RTI::ObjectID theObject
 const RTI::AttributeHandleSet& theAttributes
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotKnown,
  RTI::FederateInternalError
 )
```

ARGUMENTS

theObject

object ID of the object for whom an attribute-value update is requested.

theAttributes

set of attributes for which an update is requested; these must be attributes whose ownership tokens are held by the federate. The caller maintains ownership of the storage associated with this set; the federate ambassador should make a copy if it wishes to retain this information after the completion of the call.

DESCRIPTION

This callback is invoked to notify the federate that an attribute-update has been requested by another federate via its *RTIambassador::requestClass,ObjectAttributeValueUpdate* service. (Note that one attribute-value request can trigger multiple provide attribute-value update callbacks on different federates.)

Upon receipt of such a request, the federate should update the specified object-attributes (using *RTIambassador::updateAttributeValues*) as soon as possible. (Keep in mind that this may not be done from inside the *FederateAmbassador::provideAttributeValueUpdate* callback as this would result in a concurrent access violation.)

RETURN VALUES

A non-exceptional return indicates that the federate understands the attribute value update request and intends to update the specified object-attributes.

An exceptional return will cause an entry to be made in the federate's RTI log file; the federate is still responsible for updating the requested attributes.

EXCEPTIONS

RTI::ObjectNotKnown - The object ID is not valid in the context of the current federation execution or the object is not known by the federate.

RTI::AttributeNotKnown - One or more of the attribute handles is not valid in the context of the specified object or the federate does not hold the attribute's ownership token.

RTI::FederateInternalError - An error internal to the federate has occurred.

SEE ALSO

RTIambassador::requestObjectAttributeValueUpdate, *RTIambassador::updateAttributeValues*

3.3.16 Retract

NAME

retract - cancel an update, interaction, or deletion previously scheduled by the federate

HLA INTERFACE SPECIFICATION SERVICE

4.16 - *Object Management* (federate initiated)

SYNOPSIS

```
void
RTIambassador::retract (
  RTI::EventRetractionHandle theHandle
)
 throw (
  RTI::InvalidRetractionHandle,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theHandle

event-retraction handle (as obtained from RTIambassador::updateAttributeValues, sendInteraction, deleteObject) of the event to unschedule.

DESCRIPTION

The federate can utilize this service to withdraw an update, interaction, or deletion it has previously scheduled (or, in 1.0, has been scheduled by another federate.) A successful invocation will result in the issuance of an event-retraction message to every federate in the federation execution. If the specified event is currently queued for delivery to a given remote federate, it is removed from its queue. If the specified event has been recently delivered to the federate (1.0 remembers the last 50,000 events delivered), the federate's *FederateAmbassador::reflectRetraction* callback is invoked and the federate is responsible for rolling back its state as appropriate.

While the RTI event retraction services don't do very much in and of themselves, they provide a cornerstone upon which optimistic simulations can be built using such techniques as "anti-messages."

RETURN VALUES

A non-exceptional return indicates that the other federates in the federation execution have been advised to cancel the event specified by the retraction handle.

EXCEPTIONS

RTI::InvalidRetractionHandle - The event-retraction handle does not correspond to an event previously scheduled by the federate. (Not thrown in 1.0.)

RTI::FederateNotExecutionMember - The RTI ambassador is not currently

associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::reflectRetraction, IRTIambassador::updateAttributeValues, RTIambassaador::sendInteraction, RTIambassador::deleteObject

3.3.17 Reflect Retraction +

NAME

reflectRetraction - inform the federate that a previously-delivered attribute-value update, interaction, or object deletion notification has been cancelled

HLA INTERFACE SPECIFICATION SERVICE

4.17 - *Object Management* (RTI initiated)

SYNOPSIS

```
virtual
void
FederateAmbassador::reflectRetraction (
  RTI::EventRetractionHandle theHandle
)
 throw (
   RTI::EventNotKnown,
   RTI::FederateInternalError
 )
```

ARGUMENTS

theHandle

event retraction handle of the event to retract (should correspond to an event-retraction handle previously passed to *FederateAmbassador::discoverObject,reflectAttributeValues,receiveInteraction,removeObject*.)

DESCRIPTION

This callback is invoked when the federate's time manager receives an event retraction request for an event that has already been delivered to the federate. The retraction request is the result of an *RTIambassador::retract* invocation by another federate (in 1.0, not necessarily the federate issuing the event.) The federate is responsible for rolling back its state as is necessary to accommodate the cancellation.

The event-retraction facilities in 1.0 are very open-ended, allowing a variety of optimistic simulation techniques such as "anti-messages" to be built on top of them.

RETURN VALUES

A non-exceptional return indicates that the federate recognizes the event retraction handle and will take the necessary steps to roll-back its federate state.

An exceptional return will cause an entry in the federate's RTI log file; the event is still assumed to have been successfully retracted.

EXCEPTIONS

RTI::EventNotKnown - The specified event retraction handle does not correspond to an event that has previously been delivered to the federation.

RTI::FederateInternalError - An error internal to the federate has occurred.

SEE ALSO

RTIambassador::retract, FederateAmbassador::discoverObject,

FederateAmbassador::reflectAttributeValues, *FederateAmbassador::receiveInteraction*, *FederateAmbassador::removeObject*

3.4 Ownership Management

Table 3-26: Ownership Management Services

Section	Service Title	Service Implemented
5.1	Request Attribute Ownership Divestiture	Yes
5.2	Request Attribute Ownership Assumption †	Yes
5.3	Attribute Ownership Divestiture Notification †	Yes
5.4	Attribute Ownership Acquisition Notification †	Yes
5.5	Request Attribute Ownership Acquisition	Yes
5.6	Request Attribute Ownership Release †	Yes
5.7	Query Attribute Ownership	Yes

3.4.1 Request Attribute Ownership Divestiture

NAME

requestAttributeOwnershipDivestiture - inform the RTI that the federate wishes to relinquish ownership of a specified set of object attributes and solicit bids to assume ownership of said attributes

HLA INTERFACE SPECIFICATION SERVICE

5.1 - *Ownership Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

enum RTI::OwnershipDivestitureCondition      NEGOTIATED = 1,
UNCONDITIONAL ;

void
requestAttributeOwnershipDivestiture (
 RTI::ObjectID theObject
 const RTI::AttributeHandleSet& theAttributes
 RTI::OwnershipDivestitureCondition theCondition
 const RTI::UserSuppliedTag theTag
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotDefined,
  RTI::AttributeNotOwned,
  RTI::InvalidDivestitureCondition,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

A variation allows the divesting federate to designate which federates may submit bids for attribute ownership:

```
void
requestAttributeOwnershipDivestiture (
 RTI::ObjectID theObject
 const RTI::AttributeHandleSet& theAttributes
 RTI::OwnershipDivestitureCondition theCondition
 const RTI::UserSuppliedTag theTag
 const RTI::FederateHandleSet& theCandidates
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotDefined,
  RTI::AttributeNotOwned,
  RTI::InvalidDivestitureCondition,
  RTI::FederateDoesNotExist,
  RTI::FederateNotExecutionMember,
```

RTI::ConcurrentAccessAttempted,
RTI::SaveInProgress,
RTI::RestoreInProgress,
RTI::RTIinternalError
)

ARGUMENTS

theObject
object whose attributes the federate wishes to divest ownership

theAttributes
set of attributes that the federate wishes to divest ownership of. The caller is responsible for freeing the memory associated with the set and may do so at any time after the completion of the call.

theCondition
enumerated value indicating whether the federate wishes to be relieved of attribute publication responsibility even in the absence of an accepting federate.

theTag
string value that can contain a description of the reason for the divestiture request or any other data that is meaningful for a particular federation. The caller is responsible for freeing the storage associated with this string and may do so at any time after the completion of the call.

theCandidates
set of federates that are to be offered ownership of the attributes being divested. The caller is responsible for freeing the storage associated with this set and may do so at any time after the completion of the call.

DESCRIPTION

The federate uses this method to request to be relieved of attribute publication responsibility. The federate must hold the ownership token for all attributes it attempts to divest; multiple negotiated divestiture requests may be made for the same object attribute in the absence of an acquiring federate.

The request sent out by this method will result in the invocation of the *FederateAmbassador::requestAttributeOwnershipAssumption* method in each of the candidate federates (or all federates join in the execution if no candidate federates were specified.) This may result in one or more of these federates making a bid for some or all of the offered attributes; the divesting federate's object manager will transfer the ownership token for a given attribute to the submitter of the first bid (or attribute acquisition request) received for the attribute.

If the federate specifies a negotiated divestiture, it will retain ownership of a given attribute until the receipt of a notification that another federate is willing to assume ownership (this notification can consist of an ownership bid made in response to the attribute assumption request or an attribute acquisition request made independently.) At this point, the federate's object manager will send an attribute assumption notification to the assuming federate and inform the divesting federate, via a *FederateAmbassador::attributeOwnershipDivestitureNotification* callback, that it has been relieved of ownership of the attribute.

If the federate specifies an unconditional divestiture, it is immediately relieved of ownership of all divested attributes and is notified of this via a *IFederateAmbassador::attributeOwnershipDivestitureNotification* callback (this callback does not occur synchronously with respect to the attribute divestiture request, but is queued up for future processing by the *RTIambassador::tick* service.) At this point, the ownership tokens are not held by any federate (technically, the tokens still reside in the object manager of the divesting federate) and will be transferred to the first federate expressing interest (i.e. responding to the attribute assumption request or submitting an independent request for attribute acquisition.)

Note that the candidate set only defines which federates are explicitly offered ownership of the tokens via the *FederateAmbassador::requestAttributeOwnershipAssumption* method; this does not prevent non-candidate federates from assuming ownership of the tokens by making an attribute acquisition request via the *RTIambassador::requestAttributeOwnershipAcquisition* service.

There is no negative acknowledgement of a negotiated attribute divestiture request; if the federate receives no positive response within a reasonable amount of time it may wish to issue another attribute divestiture request.

Multiple ownership divestiture notifications may result from a single divestiture request, as different federates may assume ownership of different subsets of the offered attributes.

RETURN VALUES

A non-exceptional return indicates that the candidate federates (or all federates, if no candidate federates were specified) have been offered ownership of the specified attributes and that the calling federate's object manager has been instructed to transfer the attribute ownership tokens to the first federate willing to assume them. If an unconditional divestiture was specified, an *attributeOwnershipDivestitureNotification* has been queued for delivery to the federate ambassador on a subsequent *RTIambassador::tick* (if a negotiated divestiture was specified, these notifications are given only after a given attribute has been divested.)

EXCEPTIONS

RTI::ObjectNotKnown - The specified object handle does not correspond to an object known by the federate's object manager.

RTI::AttributeNotDefined - One or more of the attribute handles is not valid in the context of the specified object.

RTI::AttributeNotOwned - One or more of the ownership tokens for the specified attributes is not held by the federate.

RTI::InvalidDivestitureCondition - The specified divestiture condition was not a recognized enumerated value.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI

ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::requestAttributeOwnershipAssumption, *FederateAmbassador::attributeOwnershipDivestitureNotification*, *RTIambassador::requestAttributeOwnershipAcquisition*, *RTIambassador::queryAttributeOwnership*, *RTI::FederateHandleSet*, *RTI::AttributeHandleSet*

3.4.2 Request Attribute Ownership Assumption +

NAME

requestAttributeOwnershipAssumption - informs the federate of another federate's desire to divest ownership of a set of attributes and allows the federate to submit a bid on some or all of the offered attributes

HLA INTERFACE SPECIFICATION SERVICE

5.2 - *Ownership Management* (RTI initiated)

SYNOPSIS

```
#include <RTI.hh>

virtual
RTI::AttributeHandleSet&
FederateAmbassador::requestAttributeOwnershipAssumption (
 RTI::ObjectID theObject
 const RTI::AttributeHandleSet& offeredAttributes
 const RTI::UserSuppliedTag theTag
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeAlreadyOwned,
  RTI::FederateInternalError
 )
```

A variation on this method does not include the user-supplied-tag argument:

```
virtual
RTI::AttributeHandleSet&FederateAmbassador::requestAttributeOwne
rshipAssumption (
 RTI::ObjectID theObject
 const RTI::AttributeHandleSet& offeredAttributes
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeAlreadyOwned,
  RTI::FederateInternalError
 )
```

ARGUMENTS

theObject

handle of the object whose attribute ownership tokens are being offered.

offeredAttributes

set of attributes whose ownership tokens are being offered. The caller maintains ownership of the storage space associated with this set; the federate ambassador should create a copy if it wishes to retain the values after the completion of the call.

theTag

string provided as an argument to the *RTIambassador::requestAttributeOwnershipDivestiture* method invocation

that initiated the attribute assumption request. This string may contain a description of the reason for the divestiture request or any other data this is meaningful for a particular federation. The caller maintains ownership of the storage space associated with the string; the federate ambassador should create a copy if it wishes to retain this information after the completion of the call.

DESCRIPTION

This callback method is invoked upon the receipt of a request from another federate to assume ownership of a set of attributes. This request may be the result of an explicit action taken by the federate (i.e. a call to the *RTIambassador::requestAttributeOwnershipDivestiture* service method) or an implicit divestiture of ownership tokens due to an unsubscription or resignation.

To be eligible for assumption, an attribute must be published and subscribed by the federate; ineligible attributes are automatically filtered out by the object manager and will not be presented to the *FederateAmbassador::requestAttributeOwnershipAssumption* method.

The federate ambassador communicates (via the return value) which attributes, if any, it is willing to assume ownership of. The return of a non-empty attribute set results in an ownership token bid being placed on the behalf of the federate. Usually, attribute tokens are transferred to the submitter of the first ownership bid received. If the federate receives ownership of any tokens, it will be notified via a *FederateAmbassador::attributeOwnershipAcquisitionNotification* callback.

There is no negative acknowledgement of an ownership token bid; no further action will occur as a result of a bid for a given attribute if the federate fails to acquire ownership of the attribute.

RETURN VALUES

The method should allocate storage space for the return value on the heap (using the *AttributeHandleSetFactory::create* service); the caller assumes responsibility for the disposal of this storage.

The return of a non-empty attribute set indicates the federate's desire to submit a bid for ownership of the contained attributes. Note that the willingness to accept ownership of an attribute does not guarantee that ownership will be granted; the federate does not assume ownership until it receives an *FederateAmbassador::attributeOwnershipAcquisitionNotification* callback. If multiple federates are submitting bids, as is often the case, it is possible that the federate will not be granted ownership of some or all of the attributes it agrees to accept.

EXCEPTIONS

RTI::ObjectNotKnown - The object ID does not correspond to an object previously discovered by the federate.

RTI::AttributeAlreadyOwned - One or more of the attributes contained in the set is already owned by the federate.

RTI::FederateInternalError - An error internal to the federate has occurred; this exception will result in an entry being made to the federate's RTI log.

SEE ALSO

RTIambassador::requestAttributeOwnershipDivestiture, FederateAmbassador::attributeOwnershipAcquisitionNotification, RTIambassador::unsubscribeObjectClassAttribute, RTIambassador::resignFederationExecution, RTI::AttributeHandleSet

3.4.3 Attribute Ownership Divestiture Notification +

NAME

attributeOwnershipDivestitureNotification - informs the federate that ownership tokens for a set of attributes have been divested; the federate is relieved of publication duties and can no longer produce updates for the divested attributes

HLA INTERFACE SPECIFICATION SERVICE

5.3 - *Ownership Management* (RTI initiated)

SYNOPSIS

```
#include <RTI.hh>

virtual
void
FederateAmbassador::attributeOwnershipDivestitureNotification (
 RTI::ObjectID theObject
 const RTI::AttributeHandleSet& releasedAttributes
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotKnown,
  RTI::FederateInternalError
 )
```

ARGUMENTS

theObject

object whose attribute have been divested.

releasedAttributes

set of attributes that the federate is being relieved of. The caller maintains ownership of the storage space for this set; the federate ambassador must make a copy if it wishes to retain this information after the completion of the call.

DESCRIPTION

This method informs the federate that it has been relieved of a set of attributes for which it has an outstanding ownership divestiture request, or has agreed to release as per a *FederateAmbassador::requestAttributeOwnershipRelease* request.

If an unconditional divestiture was requested, the federate will receive this notification immediately after the divestiture request (*RTIambassador::requestAttributeOwnershipDivestiture* will enqueue the notification in the federate's pending message list for processing by *RTIambassador::tick*.) If a negotiated divestiture was requested, this notification is not issued until the receipt of a positive response to the divestiture request or the receipt of attribute acquisition request made by another federate.

Multiple ownership divestiture notifications may result from a single divestiture request, as different federates may assume ownership of different subsets of the offered attributes.

If the federate has agreed to release attributes as per an acquisition request made by

another federate, the divestiture notification will be delivered immediately upon the return from the *FederateAmbassador::requestAttributeOwnershipRelease* method (i.e. the RTI does not wait until a subsequent *RTIambassador::tick* as is the usual policy for delivery of notifications.)

Subsequent attempts to update divested attributes will result in an *RTI::AttributeNotOwned* exception.

RETURN VALUES

A non-exceptional return indicates that the federate recognizes the specified object and attributes and will cease publication of updates for the attributes. (Even if this method raises an exception, the federate will still be relieved of ownership of the specified tokens after an entry has been made in the federate's RTI log.)

EXCEPTIONS

RTI::ObjectNotKnown - The federate has not discovered an object with the specified object ID.

RTI::AttributeNotKnown - One or more of the attributes are not valid in the context of the specified object or have not been the subject of a divestiture request.

RTI::FederateInternalError - An error internal to the federate has occurred; this exception will result in an entry being made to the federate's RTI log.

SEE ALSO

RTIambassador::requestAttributeOwnershipDivestiture, FederateAmbassador::requestAttributeOwnershipRelease, RTI::AttributeHandleSet

3.4.4 Attribute Ownership Acquisition Notification +

NAME

attributeOwnershipAcquisitionNotification - informs the federate that the ownership tokens for a set of attributes have been acquired; the federate is to immediately assume publication responsibility for the acquired attributes

HLA INTERFACE SPECIFICATION SERVICE

5.4 - *Ownership Management* (RTI initiated)

SYNOPSIS

```
virtual
voidFederateAmbassador::attributeOwnershipAcquisitionNotification
(
 RTI::ObjectID theObject
 const RTI::AttributeHandleSet& securedAttributes
)
 throw (
  RTI::ObjectNotKnown,
  AttributeNotKnown,
  FederateInternalError
 )
```

ARGUMENTS

theObject

object whose attributes have been acquired by the federate.

securedAttributes

set of attributes of the specified object that have been acquired by the federate. The caller maintains ownership of the storage space associated with the set; the federate ambassador should make a copy if it wishes to retain the information after the completion of the call.

DESCRIPTION

This method informs the federate that it has acquired ownership tokens for the given set of attributes; this acquisition may be the result of a successful ownership bid made in response to an attribute assumption request, or the receipt of a positive response to an attribute acquisition request.

Multiple acquisition notifications may result from a single acquisition request, as ownership tokens for different subsets of the set of requested attributes may reside in different federates. It is theoretically possible, though unlikely, that multiple acquisition notifications will result from a single attribute ownership bid.

RETURN VALUES

A non-exceptional return indicates that the federate recognizes the specified object and attributes and will assume ownership responsibilities of these attributes. An exceptional return will result in an error message being entered into the RTI log file; ownership of the tokens will still be acquired by the federate.

EXCEPTIONS

RTI::ObjectNotKnown - The federate has not discovered an object with the specified object ID.

AttributeNotKnown - One or more of the attribute handles are not valid within the context of the specified object, are already owned by the federate, or are not published or not subscribed by the federate.

RTI::FederateInternalError - An error internal to the federate has occurred.

SEE ALSO

RTIambassador::requestAttributeOwnershipAcquisition, FederateAmbassador::requestAttributeOwnershipAssumption, FederateAmbassador::requestAttributeOwnershipRelease, RTI::AttributeHandleSet

3.4.5 Request Attribute Ownership Acquisition

NAME

requestAttributeOwnershipDivestiture - inform the federation of the federate's desire to acquire ownership of a specified set of attributes for a specified object

HLA INTERFACE SPECIFICATION SERVICE

5.5 - *Ownership Management* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
requestAttributeOwnershipAcquisition (
 RTI::ObjectID theObject
 const RTI::AttributeHandleSet& desiredAttributes
 const RTI::UserSuppliedTag theTag
)
 throw (
  RTI::ObjectNotKnown,
  RTI::ObjectClassNotPublished,
  RTI::ObjectClassNotSubscribed,
  RTI::AttributeNotDefined,
  RTI::AttributeNotPublished,
  RTI::AttributeNotSubscribed,
  RTI::FederateOwnsAttributes,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theObject

object whose attributes the federate wishes to acquire ownership of.

theAttributes

attributes of the specified object that the federate wishes to acquire ownership of. The caller is responsible for freeing the storage associated with this set and may do so at any time after the completion of the call.

DESCRIPTION

This method informs the federation that the federate wishes to acquire ownership of the specified attributes of the specified object. If the ownership token for a given attribute exists but is not held by any federate (technically, the token is held by the object manager of some federate but not owned by the associated federate ambassador), the federate is automatically granted ownership of the attribute. If the token is held by a federate, that federate is requested to relinquish control via its *FederateAmbassador::requestAttributeOwnershipRelease* method.

If the federate successfully acquires some or all of the requested attributes, it will be notified via its *FederateAmbassador::attributeOwnershipAcquisitionNotification*

method. It is possible that a single acquisition request result in multiple acquisition notifications, as different subsets of the set of requested attributes may be held by different federates. These notifications do not occur synchronously with respect to the acquisition request method, but will occur at a later time in response to an *RTIambassador::tick* invocation.

There is no negative acknowledgement of ownership acquisition requests; if a given attribute ownership token no longer exists or if it is held by a federate that declines to relinquish control, then no further actions result from the ownership acquisition request for that attribute.

A federate must publish and subscribe a given attribute and its associated object class before attempting to acquire tokens for that attribute. (Note that here "object class" refers to the class by which the federate knows an object, which may differ from the actual class of the object.)

RETURN VALUES

A non-exceptional return indicates that a request has been made on behalf of the federate to obtain ownership of the specified attributes. It is important to note that a successful return of this method does not imply acquisition of the requested attributes, only that the request has been successfully issued.

EXCEPTIONS

RTI::ObjectNotKnown - The specified object ID is invalid in the context of the current federation execution, or the associated object is not known to the federate.

ObjectClassNotPublished - The object class of the specified object is not published by the federate.

ObjectClassNotSubscribed - The object class of the specified object is not subscribed by the federate.

AttributeNotDefined - One or more of the specified attribute handles is not valid in the context of the specified object's object class.

AttributeNotPublished - One or more of the specified attribute handles is not published by the federate.

AttributeNotSubscribed - One or more of the specified attribute handles is not subscribed by the federate.

FederateOwnsAttributes - The federate already holds ownership tokens for one or more of the attributes specified.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::attributeOwnershipAcquisitionNotification, FederateAmbassador::requestAttributeOwnershipRelease, RTIambassador::publishObjectClass, RTIambassador::subscribeObjectClassAttribute, RTIambassador::queryAttributeOwnership, RTI::AttributeHandleSet

3.4.6 Request Attribute Ownership Release +

NAME

requestAttributeOwnershipRelease - inform the federate that another federate has requested acquisition of one or more attribute ownership tokens held by the federate

HLA INTERFACE SPECIFICATION SERVICE

5.6 - *Ownership Management* (RTI initiated)

SYNOPSIS

```
#include <RTI.hh>

virtual
RTI::AttributeHandleSet
&
FederateAmbassador::requestAttributeOwnershipRelease (
 RTI::ObjectID theObject
 const RTI::AttributeHandleSet& candidateAttributes
 const RTI::UserSuppliedTag theTag
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotKnown,
  RTI::FederateInternalError
 )
```

ARGUMENTS

theObject

object whose attributes the federate is requested to release.

candidateAttributes

set of attributes that the federate is requested to release. The caller maintains ownership of the storage used by this set; if the federate wishes to retain this value beyond the termination of the call it should make a copy.

theTag

string value given as an argument to the *RTIambassador::requestAttributeOwnershipRelease* invocation that triggered the request. This string can contain a description of the reason for the acquisition request or any other value that is meaningful for a particular federation. The caller maintains ownership of the storage used by this string; if the federate wishes to retain this value beyond the termination of the call it should make a copy.

DESCRIPTION

This method informs the federate of another federate's interest in acquiring one or more of the attribute ownership tokens it holds. The federate indicates, via the return value of the method, the set of attributes for which it is willing to relinquish ownership. If the returned set is non-empty, the acquiring federate will be notified that it has assumed ownership of the tokens via its *FederateAmbassador::attributeOwnershipAcquisitionNotification* method. Unlike assumption of attributes, release of attributes is not subject to further negotiation; any valid attributes returned by this method are guaranteed to be released to the

requesting federate.

A non-empty return set will also trigger a *FederateAmbassador::attributeOwnershipDivestitureNotification* callback to the releasing federate. This notification occurs immediately after the *FederateAmbassador::requestAttributeOwnershipRelease* call (unlike most notifications, it is not queued up for future processing by *RTIambassador::tick*.)

RETURN VALUES

The method should allocate storage space for the return value on the heap (using the *AttributeHandleSetFactory::create* service); the caller assumes responsibility for the disposal of this storage.

The return of a non-empty attribute set causes ownership of the returned attributes to be divested to the requesting federate; the involved federates are notified via federate ambassador callbacks as described above.

EXCEPTIONS

RTI::ObjectNotKnown - The object ID specified does not correspond to an object that has been discovered by the federate.

AttributeNotKnown - One or more of the specified attribute handles is not valid in the context of the specified object or the attribute ownership token is not held by the federate.

RTI::FederateInternalError - An error internal to the federate has occurred; this exception will result in an entry being made to the federate's RTI log.

SEE ALSO

RTIambassador::requestAttributeOwnershipAcquisition, *FederateAmbassador::attributeOwnershipAcquisitionNotification*, *FederateAmbassador::attributeOwnershipDivestitureNotification*, *RTI::AttributeHandleSet*

3.4.7 Query Attribute Ownership

NAME

queryAttributeOwnership, informAttributeOwnership, attributeIsOwnedByFederate - determine which federate, if any, holds the attribute ownership token for a given attribute

HLA INTERFACE SPECIFICATION SERVICE

5.7 - *Ownership Management* (federate initiated)

(1.0 implements this service as a complimentary pair of methods, one federate-initiated and one RTI-initiated.)

SYNOPSIS

```
#include <RTI.hh>

void
RTIambassador::queryAttributeOwnership (
  RTI::ObjectID theObject
  RTI::AttributeHandle theAttribute
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotDefined,
  RTI::AttributeNotKnown,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )

virtual
void
FederateAmbassador::informAttributeOwnership (
  RTI::ObjectID theObject
  RTI::AttributeHandle theAttribute
  RTI::FederateHandle theOwner
)
 throw (
  RTI::ObjectNotKnown,
  RTI::AttributeNotKnown,
  RTI::FederateInternalError
 )
```

An alternate method is only able to determine if the ownership token is held by the local federate, but has the advantage of providing a response synchronously:

```
RTI::Boolean
RTIambassador::attributeIsOwnedByFederate (
  RTI::ObjectID theObject
  RTI::AttributeHandle theAttribute
)
```

throw (
RTI::ObjectNotKnown,
RTI::AttributeNotDefined,
RTI::AttributeNotKnown,
RTI::FederateNotExecutionMember,
RTI::ConcurrentAccessAttempted,
RTI::SaveInProgress,
RTI::RestoreInProgress,
RTI::RTIinternalError
)

ARGUMENTS

theObject
object whose attribute ownership status is being queried.

theAttribute
attribute whose ownership status is being queried.

theHandle
federate handle of the federate holding the attribute ownership token.

DESCRIPTION

RTIambassador::queryAttributeOwnership queries the federation in an attempt to locate the possessor of the specified attribute ownership token. If any federate holds the token, it will send back a positive acknowledgement which is delivered to the querying federate in the form of a *FederateAmbassador::informAttributeOwnership* callback. This notification does not occur synchronously with respect to the *RTIambassador::queryAttributeOwnership* call; the notification (if any) will be presented upon a subsequent invocation of the *RTIambassador::tick* service.

There is no negative acknowledgement of attribute ownership, i.e. the querying federate will not be notified if the query fails to locate the ownership token. It is therefore impossible to determine definitively that a given ownership token is not held by any federate in the federation, as the query response may take an arbitrarily long time to arrive from the owning federate.

RTIambassador::attributeIsOwnedByFederate provides a facility for the federate to quickly and synchronously determine whether a given ownership token is locally held.

Note that an attribute for which a federate has outstanding negotiated divestiture requests is still considered to be held by the federate until ownership is assumed by another federate.

RETURN VALUES

A non-exceptional return from *RTIambassador::queryAttributeOwnership* indicates that an attribute ownership query has been sent out to the federation on behalf of the federate.

A non-exceptional return from *FederateAmbassador::informAttributeOwnership* indicates that the federate understands the ownership information provided by the federation.

RTIambassador::attributeIsOwnedByFederate returns *RTI_TRUE* if the specified ownership token is held by the local federate or *RTI_FALSE* if the ownership token is held by another federate, unowned, or no longer exists.

EXCEPTIONS

RTI::ObjectNotKnown - The specified object handle does not correspond to an object known by the federate's object manager.

RTI::AttributeNotDefined - The attribute handle is not valid in the context of the specified object.

RTI::AttributeNotKnown - (Not thrown in 1.0.)

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

RTI::FederateInternalError - An error internal to the federate has occurred; this exception will result in an entry being made to the federate's RTI log.

SEE ALSO

RTIambassador::requestAttributeOwnershipAcquisition, *RTIambassador::requestAttributeOwnershipDivestiture*

3.5 Time Management

Table 3-27: Time Management Services

Section	Service Title	Service Implemented
6.1	Request Federation Time	Yes
6.2	Request LBTS	Yes
6.3	Request Federate Time	Yes
6.4	Request Minimum Next Event Time	Yes
6.5	Set Lookahead	Yes
6.6	Request Lookahead	Yes
6.7	Time Advance Request	Yes
6.8	Next Event Request	Yes
6.9	Flush Queue Request	Yes
6.10	Time Advance Grant †	Yes

3.5.1 Request Federation Time

NAME

requestFederationTime - request the current federation time

HLA INTERFACE SPECIFICATION SERVICE

6.1 - *Time Management* (federate initiated)

SYNOPSIS

```
RTI::FederationTime
RTIambassador::requestFederationTime ( )
 throw (
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

DESCRIPTION

Federation time for a given federate is defined as the minimum of the current federation lower-bound time stamp and the federate's logical time. This value represents the maximum time-stamp value that is eligible for delivery to the federation at this particular instance in time.

RETURN VALUES

The returned value is the current federation time, as perceived by the federate.

EXCEPTIONS

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::requestLBTS, RTIambassador::requestFederateTime, RTIambassador::requestMinNextEventTime, RTIambassador::timeAdvanceRequest, RTIambassador::tick

3.5.2 Request LBTS

NAME

requestLBTS - request the current effective federation lower-bound time stamp (LBTS) for the federate

HLA INTERFACE SPECIFICATION SERVICE

6.2 - *Time Management* (federate initiated)

SYNOPSIS

```
RTI::FederationTime
RTIambassador::requestLBTS ( )
 throw (
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

DESCRIPTION

The federation LBTS is defined as the minimum time-stamp such that it can be guaranteed that no federate will generate any more time-stamp-ordered events with a lower time-stamp. A time-regulating federate's LBTS is its current logical time plus its current lookahead; a non-time-regulating federate's LBTS is positive infinity, as it cannot generate any time-stamp-ordered messages. The federation LBTS is the minimum of the LBTS's of all participating federates.

Time-stamp ordered messages with a time-stamp less than LBTS may still be queued for processing, and may therefore be delivered to the federate as a result of *RTIambassador::tick* invocations; the LBTS simply guarantees that no new messages with a lower-time stamp will be queued for processing (to find out the absolute minimum time-stamp of all messages eligible for future delivery, use *RTIambassador::requestMinNextEventTime*.)

Non-time-constrained federates cannot receive TSO events, so their effective federation LBTS is infinity.

RETURN VALUES

The returned value is the current federation lower-bound time stamp.

EXCEPTIONS

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in

the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::requestFederationTime, RTIambassador::requestFederateTime, RTIambassador::requestMinNextEventTime, RTIambassador::timeAdvanceRequest, RTIambassador::tick, RTIambassador::turnRegulationOn

3.5.3 Request Federate Time

NAME

requestFederateTime - request the current federate logical time

HLA INTERFACE SPECIFICATION SERVICE

6.3 - *Time Management* (federate initiated)

SYNOPSIS

```
RTI::FederationTime
RTIambassador::requestFederateTime ( )
 throw (
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

DESCRIPTION

This service allows the federate to obtain its current logical time, i.e. the most recent time requested by the federate via *RTIambassador::timeAdvanceRequest*. If the federate is time-regulating, its logical time plus its lookahead constitutes the minimum allowable time-stamp of time-stamp-ordered messages subsequently sent by the federate. If the federate is time-constrained, the logical time represents the maximum time-stamp of time-stamp-ordered events that will be delivered to the federate prior to the next time-advance request.

RETURN VALUES

The returned value is the current federate logical time.

EXCEPTIONS

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::requestFederationTime, RTIambassador::requestLBTS, RTIambassador::requestMinNextEventTime, RTIambassador::timeAdvanceRequest, RTIambassador::tick, RTIambassador::turnRegulationOn

3.5.4 Request Minimum Next Event Time

NAME

requestMinimumNextEventTime - request the minimum possible time-stamp of the earliest time-stamp-ordered event that will ever be delivered in the federation's future

HLA INTERFACE SPECIFICATION SERVICE

6.4 - *Time Management* (federate initiated)

SYNOPSIS

```
RTI::FederationTime
RTIambassador::requestMinNextEventTime ( )
 throw (
   RTI::FederateNotExecutionMember,
   RTI::ConcurrentAccessAttempted,
   RTI::SaveInProgress,
   RTI::RestoreInProgress,
   RTI::RTIinternalError
 )
```

DESCRIPTION

The minimum next event time is defined as the largest time-stamp such that RTI can guarantee that no time-stamp-ordered (TSO) events will be delivered to the federate with a smaller time-stamp value. This is defined as the minimum of the federation lower-bound time stamp and the time-stamp of the earliest time-stamp-ordered event (if any) in the federate's event queue. Note that in the case of a non-constrained federate, this is always infinity (i.e. no TSO events and an infinite LBTS.) A time advance grant can never be made to a federation time greater than the minimum next event time.

RETURN VALUES

The returned value is the current minimum next event time for the federate.

EXCEPTIONS

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::requestFederationTime, RTIambassador::requestLBTS,

RTIambassador::requestFederateTime, RTIambassador::timeAdvanceRequest, RTIambassador::tick, RTIambassador::setTimeConstrained

3.5.5 Set Lookahead

NAME

setLookahead - redefine the lookahead window for the federate

HLA INTERFACE SPECIFICATION SERVICE

6.5 - *Time Management* (federate initiated)

SYNOPSIS

```
void
RTIambassador::setLookahead (
  RTI::FederationTimeDelta theLookahead
)
 throw (
  RTI::InvalidLookahead,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theLookahead

new lookahead value to use for the federate.

DESCRIPTION

This service allows the federate to dynamically modify its lookahead window, i.e. the amount of time between the federate logical time and the earliest allowable time-stamp on a time-stamp-ordered (TSO) event generated by the federate. Lookahead is only meaningful for time-regulating federates, as non-time-regulating federates do not generate TSO events. To minimize the overhead associated with synchronizing federation time advances, federates should make their lookahead window as large as is feasible.

If the specified lookahead is smaller than the current lookahead, the new lookahead does not go into effect immediately, as this would result in the federate breaking an earlier "promise" not to generate TSO events before a given federation time. In this case, the federate's actual lookahead is gradually decreased as the federate's logical time is increased (to preserve a constant value of "logical time + lookahead") until it becomes possible to use the specified lookahead value. If the specified lookahead is greater than the current federation lookahead, it goes into effect immediately.

Obviously, lookahead values must be non-negative. A federate's lookahead defaults to *EPSILON* as defined in *$RTI_HOME/include/RTItypes.h*.

Time-constrained zero-lookahead federates are an interesting "special case"; see *RTIambassador::timeAdvanceRequestAvailable* and *RTIambassador::nextEventRequestAvailable* for discussion of special considerations for such federates.

RETURN VALUES

A non-exceptional return indicates that the federate lookahead will be adjusted to the

specified value as soon as possible.

EXCEPTIONS

RTI::InvalidLookahead - The specified lookahead is less than the minimum allowable federate lookahead (zero.)

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::timeAdvanceRequest, *RTIambassador::timeAdvanceRequestAvailable*, *RTIambassador::requestLookahead*, *RTIambassador::requestLBTS*, *RTIambassador::nextEventRequest*, *RTIambassador::nextEventRequestAvailable*

3.5.6 Request Lookahead

NAME

requestLookahead - obtain the current lookahead window being used for the federate

HLA INTERFACE SPECIFICATION SERVICE

6.6 - *Time Management* (federate initiated)

SYNOPSIS

```
RTI::FederationTimeDelta
RTIambassador::requestLookahead ( )
 throw (
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

DESCRIPTION

This service allows the federate to obtain the value of its effective lookahead, i.e. the time window between its logical time and the minimum allowable time-stamp of a time-stamp-ordered event generated by the federate. The effective lookahead at a given time is at least as great as the current lookahead as specified by the *RTIambassador::setLookahead* service (see the section on this service for a discussion of why this is true.)

RETURN VALUES

The return value is the current effective lookahead for the federate.

EXCEPTIONS

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::timeAdvanceRequest, RTIambassador::setLookahead, RTIambassador::requestFederationTime, RTIambassador::requestFederateTime, RTIambassador::requestLBTS

3.5.7 Time Advance Request

NAME

timeAdvanceRequest - request an advance of the logical time of the federate to a specified federation time

HLA INTERFACE SPECIFICATION SERVICE

6.7 - *Time Management* (federate initiated)

SYNOPSIS

```
void
timeAdvanceRequest (
  RTI::FederationTime theTime
)
 throw (
  RTI::TimeAdvanceAlreadyInProgress,
  RTI::FederationTimeAlreadyPassed,
  RTI::InvalidFederationTime,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

A variation is useful for zero-lookahead federates:

```
void
RTIambassador::timeAdvanceRequestAvailable (
  RTI::FederationTime theTime
)
 throw (
  RTI::TimeAdvanceAlreadyInProgress,
  RTI::FederationTimeAlreadyPassed,
  RTI::InvalidFederationTime,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theTime

federation time the federate wishes to advance its logical time to.

DESCRIPTION

These services allow the federate to request an advance in its logical time to a specified federation time and arrange to be notified when such an advance is achieved, i.e. the RTI can guarantee that all time-stamp-ordered (TSO) events delivered to the federate in the future will have a time-stamp greater than (or not less than in the case of *RTIambassador::timeAdvanceRequestAvailable*) the new federate logical time.

By requesting a time advance, the federate is agreeing to not generate any time-stamp-ordered events with a time-stamp less than the requested time plus the current federate lookahead (for non-time-regulating federates this is trivial, as such federates do not generate any time-stamp-ordered events.)

When the criteria for completion of the time-advance request have been met, the federate will be notified of such via the *FederateAmbassador::timeAdvanceGrant* callback. The federate may not make a time-advance request while any other time-advancement service is in progress, i.e. the federate is waiting on a *FederateAmbassador::timeAdvanceGrant*.

For non-time-constrained federates, time advances are trivial: by definition such federates do not receive any time-stamp-ordered events, so a time-advance grant is immediately scheduled for delivery by a subsequent invocation of the *RTIambassador::tick* service.

For time-constrained federates, a time-advance is granted when the minimum next event time (see *RTIambassador::requestMinNextEventTime*) exceeds the requested federate time. In the case of *RTIambassador::timeAdvanceRequestAvailable*, the time-advance is also granted if the minimum next event time equals the requested time and there are no queued TSO events eligible for delivery to the federate.

The *RTIambassador::timeAdvanceRequestAvailable* variation is similar to the *RTIambassador::timeAdvanceRequest* service but does not necessarily deliver all events at the requested time before issuing the time-advance grant, making it attractive for zero-lookahead federates that wish to simultaneously generate and process events at the same logical time.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully initiated the time-advancement process. TSO events from the current time through the requested time (inclusive) may now be delivered to the federate, and the federate may no longer generate TSO events with a time-stamp of less than the requested time plus the federate lookahead. The federate will receive notification of the successful completion of the time advance (as described previously) via its *FederateAmbassador::timeAdvanceGrant* callback.

EXCEPTIONS

RTI::TimeAdvanceAlreadyInProgress - A previous time advance request, next event request, or flush queue request has not yet been completed.

RTI::FederationTimeAlreadyPassed - The requested time is less than the current federate logical time.

RTI::InvalidFederationTime - Not thrown in 1.0.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::timeAdvanceGrant, RTIambassador::tick, RTIambassador::requestFederateTime, RTIambassador::requestLBTS, RTIambassador::setTimeConstrained, RTIambassador::turnRegulationOn, RTIambassador::nextEventRequest, RTIambassador::flushQueueRequest

3.5.8 Next Event Request

NAME

nextEventRequest, nextEventRequest - advance the federate's logical time to the time-stamp of the next TSO event in the federation

HLA INTERFACE SPECIFICATION SERVICE

6.8 - *Time Management* (federate initiated)

SYNOPSIS

```
void
nextEventRequest (
  RTI::FederationTime theTime
)
 throw (
  RTI::TimeAdvanceAlreadyInProgress,
  RTI::FederationTimeAlreadyPassed,
  RTI::InvalidFederationTime,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

A variation is useful for zero-lookahead federates:

```
void
nextEventRequestAvailable (
  RTI::FederationTime theTime
)
 throw (
  RTI::TimeAdvanceAlreadyInProgress,
  RTI::FederationTimeAlreadyPassed,
  RTI::InvalidFederationTime,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theTime

time to advance the federation logical time to in the absence of an intervening TSO event.

DESCRIPTION

These services allow the federate to advance in time to the time-stamp of the next TSO event occurring in the federation. A *FederateAmbassador::timeAdvanceGrant* will occur after a TSO event has been delivered or the federation lower-bound time stamp (LBTS) advances past the specified cutoff time; the grant will be to the time-stamp of the event or the specified cutoff time, respectively. In the interim, any

number of receive-ordered events and possibly some TSO events with the same time-stamp as the first TSO event will be delivered. The *RTIambassador::nextEventRequest* service defers the time-advance grant until it can be guaranteed that all TSO events at the grant-time have been delivered; the *RTIambassador::nextEventRequestAvailable* service does not, making it attractive for zero-lookahead federates that wish to simultaneously generate and process events at the same logical time.

Note that if the federate is not time-constrained, the completion criteria are trivially met (i.e. the effective federation LBTS for a non-constrained federate is always infinity), so a time advance grant to the cutoff time will be immediately scheduled for delivery by a subsequent invocation of *RTIambassador::tick*.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully announced its desire to advance in time; the federate will be notified of the successful completion of the request (as described above) via a subsequent *FederateAmbassador::timeAdvanceGrant* callback.

EXCEPTIONS

RTI::TimeAdvanceAlreadyInProgress - A previous time advance request, next event request, or flush queue request has not yet been completed.

RTI::FederationTimeAlreadyPassed - The specified federation time is less than the current logical time of the federation.

RTI::InvalidFederationTime - Not thrown in 1.0.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::timeAdvanceGrant, RTIambassador::tick, RTIambassador::requestFederateTime, RTIambassador::requestLBTS, RTIambassador::setTimeConstrained, RTIambassador::turnRegulationOn, RTIambassador::flushQueueRequest, RTIambassador::timeAdvanceRequest

3.5.9 Flush Queue Request

NAME

flushQueueRequest - flush the federate's internal event queues, violating the ordering of time-stamp-ordered messages if necessary

HLA INTERFACE SPECIFICATION SERVICE

6.9 - *Time Management* (federate initiated)

SYNOPSIS

```
void
RTIambassador::flushQueueRequest (
  RTI::FederationTime theTime
)
 throw (
  RTI::TimeAdvanceAlreadyInProgress,
  RTI::FederationTimeAlreadyPassed,
  RTI::InvalidFederationTime,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

theTime

maximum federate logical time to advance to upon completion of the flush.

DESCRIPTION

This service designates all events currently in the federate's event queue as eligible for presentation to the federate. Subsequent invocations of *RTIambassador::tick* will first process any receive-ordered events that have arrived, then will process events in the TSO queue without regard for the federation lower-bound time stamp. For any given invocation of *RTIambassador::tick*, the earliest available TSO event is processed; however, RTI may not be able to guarantee that TSO events with a lower time-stamp will not arrive in the future.

A time advance is granted when the federate has processed all TSO events that were queued at the time of the request. The grant time is the minimum of the minimum next event time and the specified cutoff time. Note that this is trivial in the case of a non-time-constrained federation, which by definition has no events in its TSO queue; in this case, a grant to the specified cutoff time will be made upon the next invocation of *RTIambassador::tick*.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully announced its desire to advance in time; the federate will be notified of the successful completion of the request via a subsequent *FederateAmbassador::timeAdvanceGrant* callback.

EXCEPTIONS

RTI::TimeAdvanceAlreadyInProgress - A previous time advance request, next event request, or flush queue request has not yet been completed.

RTI::FederationTimeAlreadyPassed - The specified federation time is less than the current logical time of the federation.

RTI::InvalidFederationTime - Not thrown in 1.0.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::timeAdvanceGrant, RTIambassador::tick, RTIambassador::requestFederateTime, RTIambassador::requestLBTS, RTIambassador::setTimeConstrained, RTIambassador::turnRegulationOn, RTIambassador::timeAdvanceRequest, RTIambassador::nextEventRequest

3.5.10 Time Advance Grant +

NAME

timeAdvanceGrant - inform the federate that a previous time advance request, flush queue request, or next event request has been completed

HLA INTERFACE SPECIFICATION SERVICE

6.10 - *Time Management* (RTI initiated)

SYNOPSIS

```
virtual
void
FederateAmbassador::timeAdvanceGrant (
  RTI::FederationTime theTime
)
 throw (
  RTI::InvalidFederationTime,
  RTI::TimeAdvanceWasNotInProgress,
  RTI::FederationTimeAlreadyPassed,
  RTI::FederateInternalError
 )
```

ARGUMENTS

theTime

time granted to, i.e. the new value of the federate's logical time.

DESCRIPTION

This callback is invoked to notify the federate of the successful completion of a time-advance service (i.e. time advance request, next event request , or flush queue request.) The specified grant time becomes the new logical time of the federation; this time is guaranteed to be no greater than the federate's minimum next event time and may be less than this time, depending on the the manner in which the grant was requested.

The receipt of a time advance grant indicates that all time-stamp-ordered (TSO) events with a time-stamp of less than the grant time that will ever occur in the federation have already been processed by the federate. Only receive-order events and TSO events occurring at exactly the grant time will be eligible for processing until the next invocation of a time-advance service.

Note that non-time-constrained federates have no concept of TSO events, so time advances are immediately granted to such federates.

RETURN VALUES

A non-exceptional return indicates that the federate understands the time advance grant notification.

An exceptional return will cause an entry to be made in the federate's RTI log file; the logical time of the federate is still considered to have been advanced.

EXCEPTIONS

RTI::InvalidFederationTime - The specified grant time is invalid.

RTI::TimeAdvanceWasNotInProgress - There is not an outstanding time advance request, next event request, or flush queue request.

RTI::FederationTimeAlreadyPassed - The specified grant time is less than the current federate logical time.

RTI::FederateInternalError - An error internal to the federate has occurred.

SEE ALSO

RTIambassador::tick, RTIambassador::requestFederateTime, RTIambassador::requestLBTS, RTIambassador::setTimeConstrained, RTIambassador::turnRegulationOn, RTIambassador::timeAdvanceRequest, RTIambassador::nextEventRequest, RTIambassador::flushQueueRequest

3.6 Data Distribution Management

Data Distribution Management services are not implemented in the F.0 version of the Run-Time Infrastructure. The DARPA funded Synthetic Theater of War (STOW) program is developing a prototype RTI specifically focusing on performance and scalability. The results of the STOW RTI DDM experiments will be incorporated at a later date.

Table 3-28: Data Distribution Management Services

Section	Service Title	Service Implemented
7.1	Create Update Region	No
7.2	Create Subscription Region	No
7.3	Associate Update Region	No
7.4	Change Thresholds †	No
7.5	Modify Region	No
7.6	Delete Region	No

3.7 RTI Support Services

Table 3-29: RTI Support Services

Section	Service Title	Service Implemented
8.1	Get Object Class Handle	Yes
8.2	Get Object Class Name	Yes
8.3	Get Attribute Handle	Yes
8.4	Get Attribute Name	Yes
8.5	Get Interaction Class Handle	Yes
8.6	Get Interaction Class Name	Yes
8.7	Get Parameter Handle	Yes
8.8	Get Parameter Name	Yes
8.9	Get Space Handle	No
8.10	Get Space Name	No
8.11	Set Time Regulating	Yes
8.12	Set Time Constrained	Yes
8.13	Tick	Yes
8.14	dequeueFIFOasynchronously	Yes

3.7.1 Get Handle and Get Name Services

NAME

getFederateName, *getFederateHandle*, *getSpaceName*, *getSpaceHandle*, *getParameterName*, *getParameterHandle*, *getInteractionClassName*, *getInteractionClassHandle*, *getAttributeName*, *getAttributeHandle*, *getObjectClassName*, *getObjectClassHandle* - convert between symbolic (string) names and RTI handles

HLA INTERFACE SPECIFICATION SERVICE

8.1-8.12 - *RTI Support Services* (federate initiated)

SYNOPSIS

Methods for conversion to/from object class handles:

```
RTI::ObjectClassHandle
RTIambassador::getObjectClassHandle (
  const RTI::ObjectClassName theName
)
 throw (
   RTI::NameNotFound,
   RTI::FederateNotExecutionMember,
   RTI::ConcurrentAccessAttempted,
   RTI::SaveInProgress,
   RTI::RestoreInProgress,
   RTI::RTIinternalError
 )

RTI::ObjectClassName
RTIambassador::getObjectClassName (
  RTI::ObjectClassHandle theHandle
)
 throw (
   RTI::ObjectClassNotDefined,
   RTI::FederateNotExecutionMember,
   RTI::ConcurrentAccessAttempted,
   RTI::SaveInProgress,
   RTI::RestoreInProgress,
   RTI::RTIinternalError
 )
```

Methods for conversion to/from attribute handles:

```
RTI::AttributeHandle
RTIambassador::getAttributeHandle (
  const RTI::AttributeName theName
  RTI::ObjectClassHandle whichClass
)
 throw (
   RTI::ObjectClassNotDefined,
   RTI::NameNotFound,
   RTI::FederateNotExecutionMember,
   RTI::ConcurrentAccessAttempted,
```

```
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )

RTI::AttributeName
RTIambassador::getAttributeName (
  RTI::AttributeHandle theHandle
  RTI::ObjectClassHandle whichClass
)
 throw (
  RTI::ObjectClassNotDefined,
  RTI::AttributeNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

Methods for conversion to/from interaction class handles:

```
RTI::InteractionClassHandle
RTIambassador::getInteractionClassHandle (
  const RTI::InteractionClassName theName
)
 throw (
  RTI::NameNotFound,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )

RTI::InteractionClassName
RTIambassador::getInteractionClassName (
  RTI::InteractionClassHandle theHandle
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

Methods for conversion to/from parameter handles:

```
RTI::ParameterHandle
RTIambassador::getParameterHandle (
  const RTI::ParameterName theName
  RTI::InteractionClassHandle whichClass
```

```
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::NameNotFound,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )

RTI::ParameterName
RTIambassador::getParameterName (
 RTI::ParameterHandle       theHandle
 RTI::InteractionClassHandle whichClass
)
 throw (
  RTI::InteractionClassNotDefined,
  RTI::InteractionParameterNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

Methods for conversion to/from space handles (not implemented in 1.0):

```
RTI::SpaceHandle
RTIambassador::getSpaceHandle (
 const RTI::SpaceName theName
)
 throw (
  RTI::NameNotFound,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError,
  RTI::UnimplementedService
 )

  RTI::SpaceName RTIambassador::getSpaceName (
      const RTI::SpaceHandle theHandle)
    throw (RTI::SpaceNotDefined,
          RTI::FederateNotExecutionMember,
          RTI::ConcurrentAccessAttempted,
          RTI::RTIinternalError,
          RTI::UnimplementedService); */
```

Methods for conversion to/from federate handles (not implemented in 1.0):

```
RTI::FederateHandle
RTIambassador::getFederateHandle (
 const RTI::FederateName theName
)
```

throw (
RTI::FederateDoesNotExist,
RTI::FederateNotExecutionMember,
RTI::ConcurrentAccessAttempted,
RTI::RTIinternalError,
RTI::UnimplementedService
)

RTI::FederateName RTIambassador::getFederateName (
RTI::FederateHandle theHandle)
throw (*RTI::FederateDoesNotExist*,
RTI::FederateNotExecutionMember,
RTI::ConcurrentAccessAttempted,
RTI::RTIinternalError,
RTI::UnimplementedService); */

ARGUMENTS

theName

string specifying the symbolic name to be converted to an RTI-defined handle. The caller is responsible for freeing the storage associated with this string and may do so at its leisure.

theHandle

RTI class, attribute, interaction, parameter, space, federate handle to be converted to a symbolic (string) name.

whichClass

object (interaction) class whose attributes (parameters) are being converted.

DESCRIPTION

These methods provide a mechanism for the federate to convert between symbolic (string) names and RTI-defined handles. The symbolic names are defined by the federate initialization file, *$RTI_CONFIG/[federation name].fed*.

These methods do not alter the internal state of RTI, therefore they may be called from inside of other *RTIambassador* methods (such as *RTIambassador::tick*.)

RETURN VALUES

If the handle or symbolic name is valid within the context of the current federation execution, these methods return the appropriate converted value.

If a method returns a string value, the caller is responsible for freeing the associated storage and may do so at its leisure.

EXCEPTIONS

RTI::AttributeNotDefined - The specified attribute handle is not valid in the context of the specified object class.

RTI::ConcurrentAccessAttempted - These methods are safe for reentrance into the RTI ambassador; this exception is not thrown.

RTI::FederateDoesNotExist - No federate with the specified federate handle or federate name is currently joined in the federation execution.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::InteractionClassNotDefined - The specified interaction class handle is not valid in the context of the current federation execution.

RTI::InteractionParameterNotDefined - The specified parameter handle is not valid in the context of the specified interaction class.

RTI::NameNotFound - The symbolic name does not correspond to a handle of the requested type.

RTI::ObjectClassNotDefined - The specified object class handle is not valid in the context of the current federation execution.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

RTI::SpaceNotDefined - The specified space handle is not valid in the context of the current federation execution.

RTI::UnimplementedService - This service is not implemented in 1.0.

SEE ALSO

RTIambassador::joinFederationExecution RTI::AttributeHandleSet, RTI::AttributeHandleValuePairSet, RTI::FederateHandleSet, RTI::ParameterHandleValuePairSet

3.7.2 Set Time Regulating

NAME

turnRegulationOn, turnRegulationOnNow, turnRegulationOff - specify whether or not the federate wishes to participate in the regulation of federation time

HLA INTERFACE SPECIFICATION SERVICE

8.11 - *RTI Support Services* (federate initiated)

SYNOPSIS

```
void
RTIambassador::turnRegulationOn ( )
 throw (
  RTI::FederationTimeAlreadyPassed,
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )

RTI::FederationTime
RTIambassador::turnRegulationOnNow ( )
 throw (
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )

void
RTIambassador::turnRegulationOff ( )
 throw (
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

DESCRIPTION

These methods allow the federate to specify whether its logical time should be considered in the determination of the federation's lower-bound time stamp (LBTS), i.e. the greatest time-stamp such that the federation can guarantee that no time-stamp ordered messages will be delivered with an earlier time-stamp.

RTIambassador::turnRegulationOnNow sets the federate's logical time to the current federation LBTS before turning time regulation on. If *RTIambassador::turnRegulationOn* is used instead, the federate must be sufficiently advanced in time that it will not generate time-stamp ordered messages that will be in the federation's past (i.e. the federate's logical time plus its lookahead must not be less than the federation LBTS.) Note that not all updates and interactions sent by

a time-regulating federate are necessarily time-stamp ordered; the ordering is determined on a per-attribute or per-interaction basis based on the definitions in the federation FED file (*$RTI_CONFIG/[federation name].fed*) or dynamically specified by the federate via *RTIambassador::changeAttributeOrderType* or *RTIambassador::changeInteractionOrderType*.

If a federate is not time-regulating, its logical time will not be considered in the determination of the federation LBTS, and all updates and interactions sent by the federate will be processed receive-order, regardless of their individual ordering policies.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully indicated its desire to participate or not participate in the regulation of federation time.

RTI::RTIambassador::turnRegulationOnNow returns the new federate logical time, i.e. the current value of the federation's lower-bound time stamp.

By default, federates are not time regulating.

EXCEPTIONS

RTI::FederationTimeAlreadyPassed - The federate cannot turn time regulation on because it would be possible for it to generate time-stamp ordered messages in the federation's past; it must advance in time, use *RTIambassador::turnRegulationOnNow* instead, or specify a greater lookahead.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::changeAttributeOrderType, RTIambassador::changeInteractionOrderType, RTIambassador::setTimeConstrained, RTIambassador::requestLBTS, RTIambassador::requestFederateTime, RTIambassador::setLookahead, RTIambassador::requestLookahead, RTIambassador::timeAdvanceRequest

3.7.3 Set Time Constrained

NAME

setTimeConstrained - specify whether or not the federate wishes to receive updates/interactions in time-stamped order

HLA INTERFACE SPECIFICATION SERVICE

8.12 - *RTI Support Services* (federate initiated)

SYNOPSIS

```
void
RTIambassador::setTimeConstrained (
 RTI::Boolean state
)
 throw (
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

ARGUMENTS

state

whether or not the federate wishes to be time constrained.

DESCRIPTION

This method allows the federate to dynamically specify whether or not RTI should take time into consideration when determining when to present events to the federation. If a federate is not time-constrained, all incoming events are processed in receive-order, i.e. they are immediately made available for processing by an *RTIambassador::tick* service call. Events only become eligible for presentation to a time constrained federate when it can be guaranteed that no time-stamp-ordered events with a lower time-stamp will be received. This ordering only applies to events that are designated by the sender as being time-stamp-ordered; events designated as receive-ordered will always be made eligible for presentation immediately.

Turning time constraints on affects only events received subsequently; it does not affect any time-stamp-ordered events that may have already been received and placed in the receive-order queue.

Federates are non-time-constrained by default.

RETURN VALUES

A non-exceptional return indicates that the federate's time constraints have been turned on or off as requested.

EXCEPTIONS

RTI::FederationExecutionDoesNotExist - The RTI does not have a federation executive registered for the given federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the

RTIambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::changeAttributeOrderType, *RTIambassador::changeInteractionOrderType*, *RTIambassador::turnRegulationOn*, *RTIambassador::tick*, *RTIambassador::timeAdvanceRequest*

3.7.4 Tick

NAME

tick - turn control over to RTI for a brief time to do internal processing and provide one notification to the federate ambassador

HLA INTERFACE SPECIFICATION SERVICE

8.13 - *RTI Support Services* (federate initiated)

SYNOPSIS

```
RTI::Boolean
tick ( )
 throw (
  RTI::SpecifiedSaveLabelDoesNotExist,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError
 )
```

A variation allows the federate to specify the amount of (wallclock) time to be consumed by the tick:

```
RTI::Boolean
tick (
 RTI::TickTime minimum
 RTI::TickTime maximum
)
 throw (
  RTI::SpecifiedSaveLabelDoesNotExist,
  RTI::ConcurrentAccessAttempted,
  RTI::RTIinternalError
 )
```

ARGUMENTS

minimum

minimum amount of time to be consumed by the tick.

maximum

maximum amount of time to be consumed by the tick.

DESCRIPTION

Most callbacks made to the federate ambassador are stimulated by this service; most invocations of *RTIambassador::tick* will result in one notification being delivered to the federate (some may result in zero or more than one.) These notifications may be the result of messages received from remote federates or may be bookkeeping notifications queued by the local federate's RTI ambassador. Even if the federate does not expect to receive any notifications, it is important to tick RTI periodically to allow it to perform internal bookkeeping functions (e.g. draining incoming buffers, sending out periodic Management Object Model updates, etc.)

If the federate is not time-constrained, RTI events (i.e. updates, interactions, and deletions) will be delivered to the federate as soon as possible. If the federate is time-constrained, events will be delivered while a time-advancement service (e.g. *RTIambassador::nextEventRequest*) is in progress. Time-constrained federates can

also elect to process receive-ordered events outside of a time-advancement service by using the *RTIambassador::dequeueFIFOasynchronously* service.

If the federate provides a minimum and maximum time value, *RTIambassador::tick* will block for a time no less than *minimum* and no greater than *maximum* seconds. This method of suspending execution is preferable to signal-based mechanisms and sleeps, as it allows RTI to continue processing in the meantime.

RTI ambassador functions, with the exception of the handful or reentrant support functions, may not be invoked from within callbacks triggered by *RTIambassador::tick*; such an attempt will result in an *RTI::ConcurrentAccessAttempted* exception.

RETURN VALUES

A non-exceptional return indicates that RTI is most appreciative of being given the opportunity to perform its necessary functions, and has possibly delivered one or more notifications to the federate via the federate ambassador.

EXCEPTIONS

RTI::SpecifiedSaveLabelDoesNotExist - The specified save label does not correspond to an existing labelled saved state.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

FederateAmbassador::timeAdvanceGrant, RTIambassador::updateAttributeValues, FederateAmbassador::discoverObject, FederateAmbassador::removeObject, FederateAmbassador::receiveInteraction, RTIambassador::joinFederationExecution, RTIambassador::dequeueFIFOasynchronously

3.7.5 dequeueFIFOasynchronously

NAME

dequeueFIFOasynchronously - specify whether or not the federate wishes to process receive-order events when no outstanding time-advance service is in progress

HLA INTERFACE SPECIFICATION SERVICE

8.14 - *RTI Support Services* (federate initiated)

SYNOPSIS

```
#include <RTI.hh>

void
dequeueFIFOasynchronously (
 RTI::Boolean theSwitch
)
 throw (
  RTI::FederateNotExecutionMember,
  RTI::ConcurrentAccessAttempted,
  RTI::SaveInProgress,
  RTI::RestoreInProgress,
  RTI::RTIinternalError
 )
```

DESCRIPTION

This service allows the federate to specify whether or not it wishes to process receive-ordered events in the absence of an outstanding time-advance service. This is only meaningful for time-constrained federates, as non-time-constrained federates always process events as soon as possible.

A true setting will result in receive-ordered events being delivered to the federate as soon as possible in response to a *RTIambassador::tick* invocation. A false setting (the default) will result in receive-ordered events being queued until the federate initiates a time-advancement service (e.g. *RTIambassador::timeAdvanceRequest*.)

RETURN VALUES

A non-exceptional return value indicates that the federate has reset its asynchronous dequeue preference.

EXCEPTIONS

RTI::FederateNotExecutionMember - The RTI ambassador is not currently associated with a federation execution.

RTI::ConcurrentAccessAttempted - An attempt has been made to reenter the RTI ambassador; most likely caused by a call to an RTI ambassador method from inside a FederateAmbassador callback method invoked by *RTIambassador::tick*.

RTI::SaveInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::RestoreInProgress - The attempted action would have resulted in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError - An internal error has occurred in RTI; consult the federate log file for more details.

SEE ALSO

RTIambassador::setTimeConstrained, RTIambassador::tick, RTIambassador::timeAdvanceRequest

4. Programming with the RTI

This section will take you through examples of applications using the 1.0 RTI. The source code to each of the examples can be found in $RTI_HOME/demo. The Hello World and Jager applications have been developed to aid in understanding and using the 1.0 RTI.

4.1 Hello World

The Hello World application is a simple HLA federate that simulates an increase in population over time. It registers one instance of the HLA object class Country and updates its name and population attributes as they change over time. To be true to its name Hello World periodically sends an interaction of type Communication. This communication causes all receiving federates to print the message ("Hello World!") to standard output. This tutorial describes the Hello World SOM and FOM, the Federation Execution Data, how to run the federate, and the source code that relates to using the RTI. Each of the HLA services used in Hello World is listed in Table 4-4-1: HLA Services Used in Hello World.

Table 4-4-1: HLA Services Used in Hello World

Federation Management	Declaration Management	Object Management	Time Management	Ownership Management
CreateFederation	PublishObjectClass	RequestId	TimeAdvanceRequest	
JoinFederation	PublishInteractionClass	RegisterObject	TimeAdvanceGrant	
ResignFederation	SubscribeObjectClassAttribute	UpdateAttributeValues		
DestroyFederation	SubscribeInteractionClass	DiscoverObject		
	ControlUpdates	ReflectAttributeValues		
	ControlInteractions	SendInteraction		
		ReceiveInteraction		
		DeleteObject		
		RemoveObject		

4.1.1 Simulation Object Model (SOM)

The SOM describes the public data that an application can communicate to the Run-Time Infrastructure. The Hello World SOM has one object class Country and one interaction class Communication. Table 4-4-2: Hello World Object Class Structure shows the OMT Object Class Structure table for the Hello World Federation.

Table 4-4-2: Hello World Object Class Structure

Object Class Structure Table
Country (PS)

The Communication interaction is initiated by the locally simulated Country object in each federate and is received by all other federate. The OMT Interaction table is shown in Table 4-4-3: Hello World Object Interaction Table.

Table 4-4-3: Hello World Object Interaction Table

Interaction Structure	Initiating Object		Receiving Object		Interaction Parameters	Init/ Sense/ React
	Class	Affected Attributes	Class	Affected Attributes		
Communication	Country	None	Country	None	Message	

The data communicated between federates in the Hello World federation is described in Table 4-4-4: Hello World Attribute/Parameter Table. The Country object class has attributes for the name and the population of the country. The Communication interaction class has one parameter that stores the message being communicated.

Table 4-4-4: Hello World Attribute/Parameter Table

Object/Interaction	Attribute/Parameter	Data-type	Cardinality	Units	...
Country	Name	string	1	None	…
	Population	double	1	None	…
Communication	Message	string	1	None	…

As defined in the Object Model Template, the Data Structure and Enumeration tables are part of the Attribute/Parameter table. However, since the Hello World federate does not use enumerations or complex types they are not depicted here.

4.1.2 Federation Object Model (FOM)

The FOM is the subset of each SOM that will participate in a given federation. Since the Hello World federation is comprised of only Hello World federates, the FOM is the same as the Hello World SOM.

4.1.3 Federation Execution Data (FED)

The Federation Execution Data (FED) specifies the FOM object & interaction class hierarchies and the attributes and parameters at each level. The transport and ordering for object attributes and interaction classes are also specified. The FED for Hello World is shown in Figure 4-8: Hello World Federation Execution Data (FED). The FED must be consistent across an entire federation. Users should take special care to ensure that the same FED file is being used by each federate since changes in attribute/parameter or class orders will cause an inconsistent run-time enumeration of these types.

```
(fed
    (objects
        (class Country
            (attribute Name             FED_BEST_EFFORT   FED_RECEIVE)
            (attribute Population       FED_BEST_EFFORT   FED_RECEIVE)
        )
```

```
    )
    (interactions
        (class Communication            FED_RELIABLE     FED_RECEIVE
            (parameter Message)
        )
    )
)
```

Figure 4-8: Hello World Federation Execution Data (FED)

Note: The MOM object and interaction classes have been omitted from this figure for simplicity. MOM classes must be in the FED file; otherwise, an exception will occur.

4.1.4 Running the Application

The Hello World application takes two command line arguments that specify the name of the federate's locally simulated Country object and the initial value of the Country's population attribute. In addition, a third argument can be provided to limit the number of cycles the simulation event loop will perform. See Figure 4-9: Hello World: Sample Output of the Application.

```
> helloWorld
usage: helloWorld <Country Name> <Initial Population> [<Number of Cycles>]

> helloWorld US 100 20

helloWorld: Federate Handle = 1
Start updates for unknown class: 3
Start interaction for unknown class: 4
Turning Country.Name Updates ON.
Turning Country.Population Updates ON.
Turning Communication Interactions ON.
Country[0] Name: US Population: 100
Country[0] Name: US Population: 101
Country[0] Name: US Population: 102.01
Country[0] Name: US Population: 103.03
Country[0] Name: US Population: 104.06
Country[0] Name: US Population: 105.101
Country[0] Name: US Population: 106.152
Country[0] Name: US Population: 107.214
Country[0] Name: US Population: 108.286
Country[0] Name: US Population: 109.369
Country[0] Name: US Population: 110.462
Country[0] Name: US Population: 111.567
Country[0] Name: US Population: 112.683
Country[0] Name: US Population: 113.809
Country[0] Name: US Population: 114.947
Country[0] Name: US Population: 116.097
Country[0] Name: US Population: 117.258
Country[0] Name: US Population: 118.43
Country[0] Name: US Population: 119.615
Country[0] Name: US Population: 120.811
Exiting helloWorld.
```

Figure 4-9: Hello World: Sample Output of the Application

4.1.5 Stepping Through the Application

This section will take you through the Hello World C++ source code which is comprised of three source (.cc) files and two header (.hh) files. Hello World defines one federate specific class that contains the majority of the code to model and maintain the state of all instances of Country class. This class is defined in Country.hh and implemented in Country.cc. The FederateAmbassador has been sub-classed to provide the RTI with a mechanism to invoke RTI initiated services on the HelloWorld federate. This class is defined in HwFederateAmbassador.hh and is implemented in HwFederateAmbassador.cc. The file helloWorld.cc contains the main() routine and is where the federate's event loop is located. It is described in the following paragraphs.

4.1.5.1 Instantiating the RTI Objects

The RTI objects that provide the interface for the RTI and the federate must be instantiated to participate in an HLA federation execution. The RTI Ambassador is the interface the federate uses to invoke the HLA services. The Federate Ambassador is the interface the RTI uses to inform the federate of responses to requests as well as object attribute updates and interactions from remote federates. See Figure 4-10: HelloWorld: Initializing the RTI Objects for

```
try
{
   //----------------------------------------------------
   // Create RTI objects
   //
   // The federate communicates to the RTI through the RTIambassador
   // object and the RTI communicates back to the federate through
   // the FederateAmbassador object.
   //----------------------------------------------------
   RTI::RTIambassador       rtiAmb;         // libRTI provided
   HwFederateAmbassador     fedAmb;         // User-defined
   .
   .
   .
}
catch ( RTI::Exception& e )
{
   cerr << "Error:" << &e << endl;
   return -1;
}
```

Figure 4-10: HelloWorld: Initializing the RTI Objects

4.1.5.2 Creating the Federation Execution

Once the RTI objects are instantiated, a federation execution must be created that a group of federates can join to participate in a distributed simulation. This service will attempt to register the named federation execution with the RTI Executive (rtiexec). Figure 4-11: HelloWorld: Creating the Federation Execution shows the source code invoking this service. Note: The 1.0 RTI requires that the RTI executive be running on a well known host on a well known port. The 1.0 library (libRTI) consults the RTI.rid file located in the directory specified by the RTI_CONFIG environment variable to determine the name of the host the RTI executive process runs on and the port it communicates through.

When a federate creates the federation execution, the $RTI_HOME/bin/fedex.sh process is executed on the localhost. The RTI_HOME environment variable must be set to the root of the RTI 1.0 distribution tree in the federate environment to allow the federate to execute the fedex process.

Note: Each federation execution must have a unique name. This will be apparent when the Join Federation Execution service is viewed.

```
try
{
   //----------------------------------------------------
   // A successful createFederationExecution will cause
   // the fedex process to be executed on this machine.
   //----------------------------------------------------
   rtiAmb.createFederationExecution( fedExecName );
}
catch ( RTI::FederationExecutionAlreadyExists& e )
{
   cerr << "Note: Federation execution already exists." << &e << endl;
}
catch ( RTI::Exception& e )
{
   cerr << "Error:" << &e << endl;
}
```

Figure 4-11: HelloWorld: Creating the Federation Execution

4.1.5.3 Joining the Federation Execution

A federate must join an existing federation execution to participate in the distributed simulation. Invocation of this service will cause the RTIambassador to establish communication channels with the named federation execution. catch (RTI::FederateAlreadyExecutionMember& e)

```
{
  cerr << "Error: " << myCountry->GetName()
```

```
            << " already exists in the Federation Execution "
            << fedExecName << "." << endl;
      cerr << &e << endl;
      return -1;
   }
   catch (RTI::FederationExecutionDoesNotExist& e)
   {
      cerr << "Error: " << fedExecName << " Federation Execution "
           << "does not exists."<< endl;
      cerr << &e << endl;
      return -1;
   }
   catch (RTI::CouldNotOpenFED& e)
   {
      cerr << "Error: The FED file $RTI_CONFIG/" << fedExecName << ".fed"
           << "could not be opened."
           << endl;
      cerr << &e << endl;
      return -1;
   }
   catch (RTI::ErrorReadingFED& e)
   {
      cerr << "Error: The FED file $RTI_CONFIG/" << fedExecName << ".fed"
```

```
            << "can not be properly read - please check the format."
            << endl;
      cerr << &e << endl;
      return -1;
   }
   catch ( RTI::Exception& e )
   {
      cerr << "Error:" << &e << endl;
   }
```

Figure 4-12: HelloWorld: Joining a Federation Execution shows the source code that invokes this service.

Note: A <Federation Execution Name>.fed file must exist in the directory specified by the RTI_CONFIG environment variable. This file contains FOM information giving the object and interaction class structures and attribute/parameter names. Additionally, default attribute and interaction transport and ordering information is given. For a more complete description of the FED, see 2.4.2 Federation Execution Data (FED).

```
   try
   {
      federateId = rtiAmb.joinFederationExecution( myCountry->GetName(),
                                                   fedExecName,
                                                   &fedAmb);
   }
   catch (RTI::FederateAlreadyExecutionMember& e)
   {
      cerr << "Error: " << myCountry->GetName()
           << " already exists in the Federation Execution "
           << fedExecName << "." << endl;
      cerr << &e << endl;
      return -1;
   }
   catch (RTI::FederationExecutionDoesNotExist& e)
   {
      cerr << "Error: " << fedExecName << " Federation Execution "
           << "does not exists."<< endl;
      cerr << &e << endl;
      return -1;
   }
   catch (RTI::CouldNotOpenFED& e)
   {
      cerr << "Error: The FED file $RTI_CONFIG/" << fedExecName << ".fed"
           << "could not be opened."
           << endl;
      cerr << &e << endl;
      return -1;
   }
   catch (RTI::ErrorReadingFED& e)
   {
      cerr << "Error: The FED file $RTI_CONFIG/" << fedExecName << ".fed"
           << "can not be properly read - please check the format."
           << endl;
      cerr << &e << endl;
      return -1;
   }
   catch ( RTI::Exception& e )
   {
      cerr << "Error:" << &e << endl;
```

```
    }
```

Figure 4-12: HelloWorld: Joining a Federation Execution

4.1.5.4 Setting Time Management

This version of Hello World allows each HelloWorld federate to progress in time as fast as it possibly can. The setTimeConstrained() service toggles whether or not the federate's time advancement is constrained by other federates' time and TSO queue. The turnRegulationOff() service informs the RTI that the federate's time and TSO queue do not have to be considered by other federates' time advancement. See Figure 4-13: HelloWorld: Setting Time Management.

```
    //-----------------------------------------------------
    // Set the Time Management parameters:
    // This version of HelloWorld operates under wall-clock
    // time (under its own control). This means that it should
    // not be constrained or regulating.
    //-----------------------------------------------------
    try
    {
       rtiAmb.setTimeConstrained( RTI::RTI_FALSE );
       rtiAmb.turnRegulationOff();
    }
    catch ( RTI::Exception& e )
    {
       cerr << "Error:" << &e << endl;
    }
```

Figure 4-13: HelloWorld: Setting Time Management

4.1.5.5 Run-Time Type Identification

1.0 does not know anything about a federate's data; therefore, it uses a Meta Object Protocol MOP to enumerate the types at run-time. Since the handles assigned to object and interaction classes, attributes, and parameters are defined at run-time, the application developer must use the RTI ancillary services to retrieve these run-time handles.

The Country::Init() method queries the RTI for the handles assigned to its object and interaction classes, attributes, and parameters. The 1.0 RTI will consistently generate the same handles for the same FED input file; however, this should not be relied upon during coding. (This means do not hard code values for the ObjectClassHandle, AttributeHandle, InteractionClassHandle, or ParameterHandle.) Figure 4-14: HelloWorld: Run-Time Type Identification Example shows the use of the RTI services that query the RTTI values for FED data.

Note: There are better ways to perform the mapping between the RTI RTTI and the simulation's compile time types. See the Jager tutorial for one approach.

```
void Country::Init( RTI::RTIambassador* rtiAmb )
{
   ms_rtiAmb = rtiAmb;

   if ( ms_rtiAmb )
```

```
   {
      //------------------------------------------------------
      // Get the RTTI (Meta-Object Protocol MOP) handles
      //
      // Since the 1.0 RTI does not know anything about your data
      // and thus uses Run-Time Type Identification we must ask the
      // RTI what to call each of our data types.
      //------------------------------------------------------
      ms_countryTypeId  = ms_rtiAmb->getObjectClassHandle(ms_countryTypeStr);
      ms_nameTypeId     = ms_rtiAmb->getAttributeHandle( ms_nameTypeStr,
                                                         ms_countryTypeId);
      ms_popTypeId      = ms_rtiAmb->getAttributeHandle( ms_popTypeStr,
                                                         ms_countryTypeId);
   }
}
```

Figure 4-14: HelloWorld: Run-Time Type Identification Example

4.1.5.6 Publishing and Subscribing to Classes of Data

The federate needs to tell the RTI the types of object class attributes and interaction classes it can produce and would like to receive. In the Hello World program all data types that are published are also instantiated and updated.

Note: Each time an object or interaction class is subscribed or published it replaces the subscription/publication for that class. In most reasonable applications the publication and subscription would be broken up into two different methods. Figure 4-15: HelloWorld: Publication and Subscription Example shows the usage of the RTI publication and subscription services.

```
void Country::PublishAndSubscribe()
{
   if ( ms_rtiAmb )
   {
      //------------------------------------------------------
      // To actually subscribe and publish we need to build
      // an AttributeHandleSet that contains a list of
      // attribute type ids (AttributeHandle).
      //------------------------------------------------------
      RTI::AttributeHandleSet *countryAttributes;
      countryAttributes = RTI::AttributeHandleSetFactory::create(2);

      countryAttributes->add( ms_nameTypeId );
      countryAttributes->add( ms_popTypeId );

      //------------------------------------------------------
      // I like to subscribe first because, in 1.0 RTI, publish
      // causes an immediate ControlUpdates service request
      // and I like to keep as few events in the queue (waiting
      // for tick) as possible for as little time as possible.
      //------------------------------------------------------
      ms_rtiAmb->subscribeObjectClassAttribute( ms_countryTypeId,
                                                *countryAttributes );
      ms_rtiAmb->publishObjectClass( ms_countryTypeId,
                                     *countryAttributes);

      countryAttributes->empty();
```

```
      delete countryAttributes;   // Deallocate the memory

      //----------------------------------------------------
      // Same as above for interactions
      //----------------------------------------------------

      // Get RTTI info
      ms_commTypeId    = ms_rtiAmb->getInteractionClassHandle( ms_commTypeStr );
      ms_commMsgTypeId = ms_rtiAmb->getParameterHandle( ms_commMsgTypeStr,
                                                        ms_commTypeId);

      // Declare my Interaction interests
      ms_rtiAmb->subscribeInteractionClass( ms_commTypeId );
      ms_rtiAmb->publishInteractionClass( ms_commTypeId );

   }
}
```

Figure 4-15: HelloWorld: Publication and Subscription Example

4.1.5.7 Instantiating HLA Objects

Each Hello World federate creates one HLA object instance of class Country and registers it with the RTI. The Country::Country() constructor method does not perform the id request and registerObject request since Country objects are also instantiated upon discovery. Figure 4-16: HelloWorld: Instantiation of HLA Objects shows the usage of the Request ID and Register Object services.

```
void Country::Register()
{
   if ( m_rtiAmb )
   {
      //----------------------------------------------------
      // Instantiate my country object then register it with
      // the RTI.  Registering an object with the RTI allows
      // the object to be discovered by other federates in the
      // federation execution.
      //
      // Note: Discovery happens after an object is registered
      //       and the subscribed attributes are updated.
      //----------------------------------------------------
      RTI::ObjectIDcount numObjects(1);

      m_rtiAmb->requestID( numObjects, m_instanceId, m_instanceId );
      m_rtiAmb->registerObject( this->GetCountryRtiId(),
                                m_instanceId );
   }
}
```

Figure 4-16: HelloWorld: Instantiation of HLA Objects

4.1.5.8 Simulation Event Loop

The Hello World simulation cycles through the event loop the number of times specified in the command line arguments. In each cycle the federate requests a time advance, ticks the RTI until the time advance is

granted, and then prints the current state of all known Country objects and updates the local Country object's state based on the time advance granted.

4.1.5.8.1 Advancing Time

The HelloWorld federate uses the timeAdvanceRequest() service. This service will provide a full grant to the requested time. Figure 4-17: HelloWorld: Time Advance Request Example shows the usage for the timeAdvanceRequest() service.

```
int counter = 0;

while ( counter++ < numerOfTicks )
{
   //----------------------------------------------------
   //
   // Advance to next time step (next year), then calculate the
   // new population.
   //
   // The RTI will asynchronously grant a time advance to us and
   // it may or may not be the entire time step we asked for.
   // This depends on the ordering specified in the RID and the
   // time advance service used.  Since this version of
   // Hello World uses timeAdvanceGrant it will be the entire
   // time step we asked for.  Other time advance methods such
   // nextEventRequest may give incremental grants.
   //----------------------------------------------------
   try
   {
      timeAdvGrant = RTI::RTI_FALSE;
      rtiAmb.timeAdvanceRequest(currentTime + timeStep);
   }
   catch ( RTI::Exception& e )
   {
      cerr << "Error:" << &e << endl;
   }
```

Figure 4-17: HelloWorld: Time Advance Request Example

4.1.5.8.2 Ticking the RTI

Invoking the RTIambassador::tick() method is how the federate turns control over to the RTI to perform internal synchronization operations and to invoke the RTI initiated services on the FederateAmbassador. This section has several subsections that describe the HwFederateAmbassador methods that are implemented in the Hello World application.

The RTIambassador::tick() method processes the next event in the RTI's internal queue and returns RTI::RTI_TRUE if additional events exist that need to be processed. Figure 4-18: HelloWorld: Providing Control to the RTI shows the RTIambassdor::tick() method being used such that all available events are processed (not just one).

```
   while (!timeAdvGrant)
   {
      //----------------------------------------------------
```

```
            //
            // Tick will turn control over to the RTI so that it can
            // process an event.  This will cause an invocation of one
            // of the federateAmbassadorServices methods.
            //
            // Be sure not to invoke the RTIambassadorServices from the
            // federateAmbassadorServices; otherwise, a ConcurrentAccess
            // exception will be thrown.
            //
            //-------------------------------------------------------
            int eventsToProcess = RTI::RTI_TRUE;

            while ( eventsToProcess )
            {
               eventsToProcess = rtiAmb.tick();
            }
         }
```

Figure 4-18: HelloWorld: Providing Control to the RTI

4.1.5.8.2.1 Control Updates

The Control Updates service allows the RTI to inform the federate when an object class the federate published is needed or is not needed by the federation. This allows the federate to only send attribute updates to the RTI when the federation requires them.

When the RTI invokes this service on the Hello World federate, the application sets a flag for each attribute to the appropriate value. During the Country object's update method, these flags are checked to make sure the federation needs the data. Figure 4-19: HelloWorld: Control Updates Example shows the usage for this service.

Note: Control Updates is hard-wired in 1.0 such that the RTI tells every federate to update all classes it publishes.

```
void HwFederateAmbassador::startUpdates(ObjectClassHandle   theClass,
                                        const AttributeHandleSet& theAttributes)
     throw (RTI::ObjectClassNotPublished,
            RTI::AttributeNotPublished,
            RTI::FederateInternalError)
{
   //----------------------------------------------------------------
   // Gets called immediately in 1.0 for all classes you publish.
   //----------------------------------------------------------------
   if ( theClass == Country::GetCountryRtiId() )
   {
      Country::SetUpdateControl( RTI::RTI_TRUE, theAttributes );
   }
   else
   {
      cerr << "Start updates for unknown class: " << theClass << endl;
   }
}

void HwFederateAmbassador::stopUpdates(ObjectClassHandle  theClass,
                                       const AttributeHandleSet& theAttributes)
```

```
     throw (RTI::ObjectClassNotPublished,
            RTI::AttributeNotPublished,
            RTI::FederateInternalError)
{
   //----------------------------------------------------------------
   // Never gets called in 1.0 but we will implement it for good form.
   //----------------------------------------------------------------
   if ( theClass == Country::GetCountryRtiId() )
   {
      Country::SetUpdateControl( RTI::RTI_FALSE, theAttributes );
   }
   else
   {
      cerr << "Stop updates for unknown class: " << theClass << endl;
   }
}

void Country::SetUpdateControl( RTI::Boolean status,
                const RTI::AttributeHandleSet& theAttrHandles )
{
   //----------------------------------------------------------------
   // Note: This is hard-wired in 1.0 - meaning all things a
   //       federate publishes will cause a start control update.
   //----------------------------------------------------------------
   RTI::AttributeHandle attrHandle;

   //----------------------------------------------------------------
   // We need to iterate through the AttributeHandleSet
   // to extract each AttributeHandle.  Based on the type
   // specified ( the value returned by getHandle() ) we need to
   // set the status of whether we should send this type of data.
   //----------------------------------------------------------------
   for ( int i = 0; i < theAttrHandles.size(); i++ )
   {
      attrHandle = theAttrHandles.getHandle( i );
      if ( attrHandle == Country::GetPopulationRtiId() )
      {
         // Turn population updates on/off
         ms_sendPopulationAttrUpdates = status;

         char *pStr = ms_sendPopulationAttrUpdates ? "ON" : "OFF";

         cout << "Turning Country.Population Updates "
              << pStr << "." << endl;
      }
      else if ( attrHandle == Country::GetNameRtiId() )
      {
         // Turn name updates on/off
         ms_sendNameAttrUpdates = status;

         char *pStr = ms_sendNameAttrUpdates ? "ON" : "OFF";

         cout << "Turning Country.Name Updates "
              << pStr << "." << endl;
      }
   }
}
```

Figure 4-19: HelloWorld: Control Updates Example

4.1.5.8.2.2 Control Interactions

The Control Interactions service allows the RTI to inform the federate when an interaction class the federate published is needed or is not needed by the federation. This allows the federate to only send interactions to the RTI when the federation requires them.

When the RTI invokes this service on the Hello World federate, the application sets a flag for each interaction class to the appropriate value. During the Country object's update method, these flags are checked to make sure the federation needs the data. Figure 4-20: HelloWorld: Control Interactions Example shows the usage for this service.

Note: Control Interactions is hard-wired in 1.0 such that the RTI tells every federate to send all interaction classes it publishes.

```
void HwFederateAmbassador::startInteractionGeneration
        (InteractionClassHandle theClass)
     throw (RTI::InteractionClassNotPublished,
            RTI::FederateInternalError)
{
   //----------------------------------------------------------------
   // Gets called immediately in 1.0 for all classes you publish.
   //----------------------------------------------------------------
   Country::SetInteractionControl( RTI::RTI_TRUE, theClass );
}

void HwFederateAmbassador::stopInteractionGeneration
        (InteractionClassHandle theClass)
     throw (RTI::InteractionClassNotPublished,
            RTI::FederateInternalError)
{
   //----------------------------------------------------------------
   // Never gets called in 1.0 but we will implement it for good form.
   //----------------------------------------------------------------
   Country::SetInteractionControl( RTI::RTI_FALSE, theClass );
}

void Country::SetInteractionControl( RTI::Boolean status,
                RTI::InteractionClassHandle theClass )
{
   if ( theClass == Country::GetCommRtiId() )
   {
      // Set a flag here so that I can tell whether I
      // need to send an interaction of this type.
      ms_sendCommInteractions = status;

      char *pStr = ms_sendCommInteractions ? "ON" : "OFF";

      cout << "Turning Communication Interactions "
           << pStr << "." << endl;
   }
   else
   {
```

```
      // If it gets this far I don't know this type of interaction
      // better let someone know.
      char *pStr = status ? "Start" : "Stop";
      cerr << pStr
           << " interaction for unknown class: " << theClass << endl;
   }
}
```

Figure 4-20: HelloWorld: Control Interactions Example

4.1.5.8.2.3 Discovering an HLA Object

When an object update occurs that meets a federates subscription, the RTI invokes the discoverObject() method on the FederateAmbassador. This method provides the federate with the object ID and the object class of the discovered object. HelloWorld instantiates an instance of class Country (after ensuring the object is of class Country). This instance is added to the Country extent (a collection of all elements of a specific type) when the Country::Country() constructor method is invoked. Figure 4-21: HelloWorld: Discovering an HLA Object shows the usage for this service.

```
void HwFederateAmbassador::discoverObject( ObjectID             theObject,
                                           ObjectClassHandle    theObjectClass,
                                           FederationTime       theTime,
                                           const UserSuppliedTag theTag,
                                           EventRetractionHandle theHandle)
    throw (RTI::CouldNotDiscover,
           RTI::ObjectClassNotKnown,
           RTI::InvalidFederationTime,
           RTI::FederateInternalError)
{
   cout << "Discovered object " << theObject << endl;

   if ( theObjectClass == Country::GetCountryRtiId() )
   {
      //----------------------------------------------------------------
      // Instantiate a local Country object to hold the state of the
      // remote object we just discovered.  This instance will get
      // stored in the static extent member data - it will be destructed
      // either when it is removed or when the application exits.
      //----------------------------------------------------------------
      Country *tmpPtr = new Country( theObject );
   }

   //----------------------------------------------------------------
   // If not Country type then don't know what to do because all I
   // know about is Country objects.
   //----------------------------------------------------------------
}
```

Figure 4-21: HelloWorld: Discovering an HLA Object

4.1.5.8.2.4 Receiving Object Attribute Updates

After an object is discovered, the RTI will provide a federate with the updates of the discovered object's attributes. In this service, the RTI does not provide the type of the object; therefore, the federate must cache

the object (type and ID) upon discovery. Figure 4-22: HelloWorld: Receiving Object Attribute Updates shows the usage of this service.

Note: The RTI encodes the attribute value buffer you provide as a bit stream since it doesn't know anything about the types of your data. When the RTI becomes available for additional platforms, this encoding will need to be supplemented by the federate or by mechanisms provided in future releases of the RTI. See 2.3.3 Data Marshaling for a more thorough discussion.

```
void HwFederateAmbassador::reflectAttributeValues
       ( ObjectID                          theObject,
         const AttributeHandleValuePairSet& theAttributes,
         FederationTime                    theTime,
         const UserSuppliedTag             theTag,
         EventRetractionHandle             theHandle )
    throw (RTI::ObjectNotKnown,
           RTI::AttributeNotKnown,
           RTI::InvalidFederationTime,
           RTI::FederateInternalError)
{
   //----------------------------------------------------------------
   // Find the Country instance this update is for.  If we can't find
   // it then I am getting data I didn't ask for.
   //----------------------------------------------------------------
   Country *pCountry = Country::Find( theObject );

   if ( pCountry )
   {
      //-------------------------------------------------------------------
      // Set the new attribute values in this country instance.
      //-------------------------------------------------------------------
      pCountry->Update( theAttributes );
   }
}
```

```
void Country::Update( const AttributeHandleValuePairSet& theAttributes )
{
   RTI::AttributeHandle attrHandle;
   unsigned long        valueLength;

   // We need to iterate through the AttributeHandleValuePairSet
   // to extract each AttributeHandleValuePair.  Based on the type
   // specified ( the value returned by getHandle() ) we need to
   // extract the data from the buffer that is returned by
   // getValue().
   for ( int i = 0; i < theAttributes.size(); i++ )
   {
      attrHandle = theAttributes.getHandle( i );
      if ( attrHandle == Country::GetPopulationRtiId() )
      {
         // We don't do any encoding when we send the data to
         // the RTI so there is no decoding here.  When we run
         // this over multiple platforms we will have a problem
         // with different endian-ness of platforms.  Either we
         // need to encode the data using something like XDR or
         // provide another mechanism.
         double population;
         theAttributes.getValue( i, (char*)&population, valueLength );
         SetPopulation( (double)population );
      }
      else if ( attrHandle == Country::GetNameRtiId() )
      {
         // Same as above goes here...
         char name[ 1024 ];
         theAttributes.getValue( i, (char*)name, valueLength );
         name[ valueLength ] = NULL;
         SetName( (const char*)name );
      }
   }
}
```

Figure 4-22: HelloWorld: Receiving Object Attribute Updates

4.1.5.8.2.5 Receiving an Interaction

When an interaction that meets a federates subscription criteria is sent, the RTI will invoke the receiveInteraction() method on the FederateAmbassador. In the Hello World application, the HwFederateAmbassador provides the interaction class handle and the set of parameters to the Country::Update() method. This method checks to ensure the type of interaction is Communication and then extracts the Message parameter to display. Figure 4-23: HelloWorld: Receiving Interactions shows the usage for this service.

```
void HwFederateAmbassador::receiveInteraction
       ( InteractionClassHandle theInteraction,
         const ParameterHandleValuePairSet& theParameters,
         FederationTime theTime,
         const UserSuppliedTag theTag,
         EventRetractionHandle theHandle)
    throw (RTI::InteractionClassNotKnown,
           RTI::InteractionParameterNotKnown,
           RTI::InvalidFederationTime,
```

```
            RTI::FederateInternalError)
{
   //  Pass the interaction off to the Country class
   // so that it can be processed.
   Country::Update( theInteraction, theParameters );
}

void Country::Update( RTI::InteractionClassHandle theInteraction,
             const RTI::ParameterHandleValuePairSet& theParameters )
{
   if ( theInteraction == Country::GetCommRtiId() )
   {
      RTI::ParameterHandle paramHandle;
      unsigned long        valueLength;

      // We need to iterate through the AttributeHandleValuePairSet
      // to extract each AttributeHandleValuePair.  Based on the type
      // specified ( the value returned by getHandle() ) we need to
      // extract the data from the buffer that is returned by
      // getValue().
      for ( int i = 0; i < theParameters.size(); i++ )
      {
         paramHandle = theParameters.getHandle( i );
         if ( paramHandle == Country::GetMessageRtiId() )
         {
            // We don't do any encoding when we send the data to
            // the RTI so there is no decoding here.  When we run
            // this over multiple platforms we will have a problem
            // with different endian-ness of platforms.  Either we
            // need to encode the data using something like XDR or
            // provide another mechanism.
            char msg[ 1024 ];
            theParameters.getValue( i, (char*)msg, valueLength );
            msg[ valueLength ] = NULL;
            cout << "Interaction: " << msg << endl;
         }
         else
         {
            // There must be an error since there should only be
            // one parameter to Communication.
            cerr << "Error: I seem to have received a parameter for "
                 << "interaction class Communication that I don't "
                 << "know about." << endl;
         }
      }
   }
   else
   {
      cerr << "Received an interaction class I don't know about." << endl;
   }
}
```

Figure 4-23: HelloWorld: Receiving Interactions

4.1.5.8.2.6 Removing HLA objects

The RTI notifies the federate when an object has either been deleted or no longer meets the federate's subscription criteria by invoking the removeObject() method on the FederateAmbassador. The Hello World application looks up the Country object based on the object ID provided by the RTI. The resulting Country object is then deleted. Figure 4-24: HelloWorld: Removing HLA Objects shows the usage of this service.

```
void HwFederateAmbassador::removeObject( ObjectID             theObject,
                                         ObjectRemovalReason  theReason,
                                         FederationTime       theTime,
                                         const UserSuppliedTag theTag,
                                         EventRetractionHandle theHandle )
     throw (RTI::ObjectNotKnown,
            RTI::InvalidFederationTime,
            RTI::FederateInternalError)
{
   //----------------------------------------------------------------
   // Call the other removeObject method since this should probably
   //  be implemented using default parameter values.
   //----------------------------------------------------------------
   this->removeObject( theObject, theReason );
}

void HwFederateAmbassador::removeObject( ObjectID             theObject,
                                         ObjectRemovalReason  theReason)
     throw (RTI::ObjectNotKnown,
            RTI::InvalidFederationTime,
            RTI::FederateInternalError)
{
   cout << "Removed object " << theObject << endl;

   Country* pCountry = Country::Find( theObject );

   if ( pCountry )
   {
      delete pCountry;
   }
}
```

Figure 4-24: HelloWorld: Removing HLA Objects

4.1.5.8.2.7 Receiving a Time Advance Grant

The timeAdvanceGrant() method on the FederateAmbassador is invoked by the RTI in reply to a request for time advancement. The Hello World application stores the time that the RTI has granted in a variable named grantTime and sets a flag informing the application that the time has changed. Figure 4-25: HelloWorld: Receiving a Time Advance Grant shows the usage of this service.

```
void HwFederateAmbassador::timeAdvanceGrant( FederationTime theTime )
     throw (RTI::InvalidFederationTime,
            RTI::TimeAdvanceWasNotInProgress,
            RTI::FederationTimeAlreadyPassed,
            RTI::FederateInternalError)
{
   grantTime = theTime;
   timeAdvGrant = RTI::RTI_TRUE;
```

```
}
```

Figure 4-25: HelloWorld: Receiving a Time Advance Grant

4.1.5.8.3 Updating state and sending data

The final step to the Hello World event loop is to update the state of the Country object by calculating the new population based on the delta time. When a Country object's member data is modified a flag is set that records that the attribute has changed. After calculation of the new state, the Country::CreateNvpSet() method is invoked to create an AttributeHandleValuePairSet that contains each of the changed attributes. The CreateNvpSet() method checks the changed flags as well as the control update flags to make sure that 1.) the data has changed and 2.) that the federation needs the data. If both of these tests pass then the attribute is included in the AttributeHandleValuePairSet. Figure 4-26: HelloWorld: Updating Country Objects and Sending Attributes and Interactions shows the usage of this service.

```
// This is the body of the simulation event loop
        //------------------------------------------------------
        // If a time advance grant occurred and we have been given
        // permission to advance in time then calculate my next state.
        //------------------------------------------------------
        if (grantTime > currentTime)
        {
           //------------------------------------------------------
           // Print state of all countries
           //------------------------------------------------------
           Country* pCountry( NULL );
           for ( int i = 0; i < Country::ms_extentCardinality; i++ )
           {
              pCountry = Country::ms_countryExtent[ i ];

              if ( pCountry )
              {
                 cout << "Country[" << i << "] " << pCountry << endl;
              }
           }

           myCountry->Update( grantTime );
           currentTime = grantTime;
        }
     } // end while

RTI::AttributeHandleValuePairSet* Country::CreateNVPSet()
{
   RTI::AttributeHandleValuePairSet* pCountryAttributes( NULL );

   // Make sure the RTI Ambassador is set.
   if ( ms_rtiAmb && hasNameChanged && hasPopulationChanged )
   {
      //------------------------------------------------------
      // Set up the data structure required to push this
      // object's state to the RTI.
      //------------------------------------------------------
      RTI::ObjectIDcount numAttributes(2);
      pCountryAttributes = RTI::AttributeSetFactory::create( numAttributes );
```

```
      if ( ( hasNameChanged == RTI::RTI_TRUE ) &&
           ( ms_sendNameAttrUpdates == RTI::RTI_TRUE ) )
      {
         // We don't do any encoding here even though the data
         // is going over the wire.  When we run this over
         // multiple platforms we will have a problem
         // with different endian-ness of platforms.  Either we
         // need to encode the data using something like XDR or
         // provide another mechanism.
         pCountryAttributes->add( this->GetNameRtiId(),
                                  (char*) this->GetName(),
                                  (strlen(this->GetName())*sizeof(char)) );
      }

      if ( ( hasPopulationChanged == RTI::RTI_TRUE ) &&
           ( ms_sendPopulationAttrUpdates == RTI::RTI_TRUE ) )
      {
         // Same goes here as above...
         pCountryAttributes->add( this->GetPopulationRtiId(),
                                  (char*) &this->GetPopulation(),
                                  sizeof(double) );
      }
   }

   // pCountryAttributes is allocated on the heap and must be
   // deallocated by the federate.
   return pCountryAttributes;
}

void Country::Update( RTI::FederationTime& newTime )
{
   //---------------------------------------------------
   // If a time advance grant occurred and we have been given
   // permission to advance in time then calculate my next state.
   //---------------------------------------------------
   double deltaTime = newTime - this->GetLastTime();

   if ( deltaTime > 0 )
   {
      SetPopulation( GetPopulation() +
                     (GetPopulation()*ms_growthRatePerSec*deltaTime) );
   }

   if ( ms_rtiAmb )
   {
      //---------------------------------------------------
      // Update state of country
      //---------------------------------------------------
      try
      {
         RTI::AttributeHandleValuePairSet* pNvpSet(this->CreateNVPSet());

         // Note: if timeAdvGrant is NULL -> SEGV:
         ms_rtiAmb->updateAttributeValues( this->GetInstanceId(),
                                           *pNvpSet,
                                           newTime, "");
         // Must free the memory
```

```
        pNvpSet->empty();
        delete pNvpSet;
     }
     catch ( RTI::Exception& e )
     {
         cerr << "Error:" << &e << endl;
     }


     // Periodically send an interaction to tell everyone Hello
     static int periodicMessage = 0;
     if ( (periodicMessage++%100) == 0 )
     {
        RTI::ParameterHandleValuePairSet* pParams( NULL );

        //------------------------------------------------------
        // Set up the data structure required to push this
        // object's state to the RTI.
        //------------------------------------------------------
        RTI::ULong numParams(1);
        pParams = RTI::ParameterSetFactory::create( numParams );

        char *pMessage = "Hello World!";

        pParams->add( this->GetMessageRtiId(),
                      (char*) pMessage,
                      (strlen(pMessage)*sizeof(char)) );
        try
        {
           ms_rtiAmb->sendInteraction( GetCommRtiId(), *pParams, newTime, "" );
        }
       catch ( RTI::Exception& e )
       {
          cerr << "Error:" << &e << endl;
       }

        // Need to free memory
        pParams->empty();
        delete pParams;
     }
   }

   // Set last time to new time
  m_lastTime = newTime;
}
```

Figure 4-26: HelloWorld: Updating Country Objects and Sending Attributes and Interactions

4.2 Jager: Another Game Exploiting the RTI (JAGER)

JAGER is a space combat game developed to demonstrate the usage of the 1.0 RTI in a more advance application. An HTML document describing the game and a tutorial on how it uses the RTI can be found in $RTI_HOME/demo/Jager/doc/tutorial.html.

5. Troubleshooting

Having troubles? Please read the following problems and resolutions.

1. Problem: Federate can not connect to rtiexec.

```
> helloWorld US 100 10
RTIexecProxy::start: Can't connect to RTIexec: Connection refused
Error:RTI::RTIinternalError : RTIexecProxy::start: Can't connect to
RTIexec: Connection refused 11000
```

Resolution: Several configuration errors can cause this problem; the dependencies include

- RTI_CONFIG environment variable,
- RTI_EXEC_HOST value in $RTI_CONFIG/RTI.rid,
- RTI_EXEC_PORT value in $RTI_CONFIG/RTI.rid, and
- proper execution of rtiexec process: rtiexec <port number>.

Either the rtiexec is not running, the rtiexec is running on the wrong port, the rtiexec is running on the wrong host, the rtiexec is running on an unreachable host, RTI_EXEC_HOST is improperly set, or RTI_EXEC_PORT is improperly set.

Check to make sure that 1) the host name and port number are correctly specified in the $RTI_CONFIG/RTI.rid file, 2) the rtiexec process was executed on the host and port specified, and 3) the host is reachable (`ping <hostname>`).

2. Problem: RTI_HOME environment variable is not set. The federate that successfully registers the federation execution with the rtiexec will attempt to fork the $RTI_HOME/bin/fedex.sh process. This attempt will fail if the variable is not set or is improperly set.

```
> helloWorld US 100 10
Error:RTI::RTIinternalError :
RTIambassador::createFederationExecution: RTI_HOME not set 13148
Error: HelloWorld Federation Execution does not exists.
RTI::FederationExecutionDoesNotExist : RTIexecImpl::getFedExByName:
not reserved 12104
```

Resolution: Set the RTI_HOME environment variable to the directory specified during installation with "/rti" appended to the end.

```
> setenv RTI_HOME $INSTALL_DIR/rti
```

3. Problem:RTI_HOME environment variable is incorrectly set. The federate that successfully registers the federation execution with the rtiexec will attempt to fork the $RTI_HOME/bin/fedex.sh process. This attempt will fail if the variable is incorrectly set.

```
> helloWorld US 100 10
Error:RTI::RTIinternalError :
RTIambassador::createFederationExecution: exec failed: No such file
or directory 13151
Error: HelloWorld Federation Execution does not exists.
```

```
RTI::FederationExecutionDoesNotExist : RTIexecImpl::getFedExByName:
not registered 12105
Error: HelloWorld Federation Execution does not exists.
RTI::FederationExecutionDoesNotExist : RTIexecImpl::getFedExByName:
not registered 12105
```

Resolution: Set the RTI_HOME environment variable to the directory specified during installation with "/rti" appended to the end.

```
> setenv RTI_HOME $INSTALL_DIR/rti
```

4. Problem:RTI_CONFIG environment variable is not set. The federate needs to read in the RTI.rid and <FederationName>.fed files that are located in the RTI_CONFIG directory.

```
> helloWorld US 100 10
RTI_CONFIG environment variable not set!!!
Error:RTI::RTIinternalError : RTI_CONFIG environment variable not set
0
```

Resolution: Set the RTI_CONFIG environment variable to the directory specified during installation with "/rti/config" appended to the end.

```
> setenv RTI_HOME $INSTALL_DIR/rti/config
```

5. Problem:RTI>rid file can not be found. RTI_CONFIG environment variable is incorrectly set. The federate needs to read in the RTI.rid and <FederationName>.fed files that are located in the RTI_CONFIG directory.

```
> helloWorld US 100 10
/home/wrong_dir/RTI.rid: Config file not found
Error:RTI::RTIinternalError : Config file not found 0
```

Resolution: Set the RTI_CONFIG environment variable to the directory specified during installation with "/rti/config" appended to the end.

```
> setenv RTI_HOME $INSTALL_DIR/rti/config
```

6. Problem:Error in syntax of MOM portion of the FED file. This generally occurs when an name is misspelled or not specified in the file.

```
> helloWorld US 100 10
Error:RTI::RTIinternalError : Error in MOM section of the FED file.
Check Syntax of the predefined MOM class. 8001
helloWorld: Federate Handle = 13340
Error:RTI::FederateNotExecutionMember : Federate not execution member
13156
Error:RTI::FederateNotExecutionMember : Federate not execution member
13156
```

Resolution: Compare the fed_example.fed file that is located in $INSTALL_DIR/rti/config against the FED file causing the exception. New releases of the RTI may add MOM classes and attributes/parameters to the MOM portion of the FED (class Manager).

7. Problem:Error in syntax of Federate portion of the FED file. This generally occurs when an name is misspelled or not specified in the file.

```
> helloWorld US 100 10
helloWorld: Federate Handle = 1
ERR RTIfedExMsgHandler::close: failed to remove handler
Error:RTI::NameNotFound : Invalid Attribute Name 5028
```

Resolution: Locate the misspelled or missing name in the FED file and fix it.